NAVAL TACTICAL DATA SYSTEM (NTDS)

technical note no. 244
AN/USQ-20 UNIT COMPUTER CHARACTERISTICS

Remington Rand Univac

DIVISION OF SPERRY RAND CORPORATION

UNIVAC PARK, ST. PAUL 16, MINNESOTA

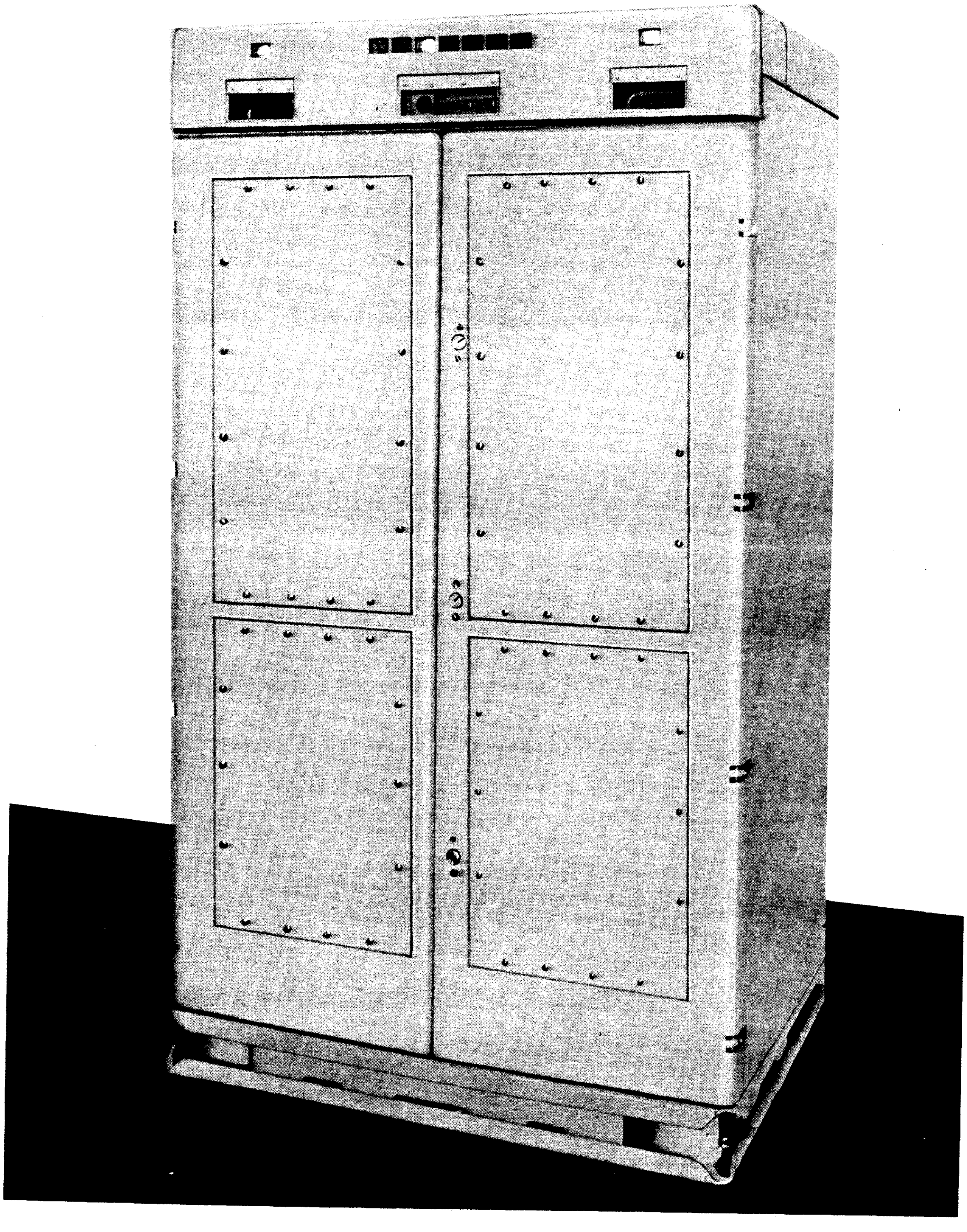
NAVAL TACTICAL DATA SYSTEM TECHNICAL NOTE NO. 244

AN/USQ-20 UNIT COMPUTER CHARACTERISTICS

PX 1343-38

Remington Rand Univac®
DIVISION OF SPERRY RAND CORPORATION
UNIVAC PARK, ST. PAUL 16, MINNESOTA

NAVY DEPARTMENT BUREAU OF SHIPS ELECTRONICS DIVISIONS
CONTRACT: NObsr 72769 NTDS NO. U-6095 10 OCTOBER 1960



CONTENTS

ILLUSTRATIONS

TABLES

TECHNICAL NOTE NO. 244

AN/USQ-20 UNIT COMPUTER CHARACTERISTICS

1. INTRODUCTION

The AN/USQ-20 Unit Computer, brain of the Naval Tactical Data System, is a general-purpose, stored-program machine capable of processing very rapidly a large quantity of complex data. Major features of the AN/USQ-20 Unit Computer include the following:

1) *Internal high-speed storage with a cycle time of 8 microseconds and a capacity of 32,768 words (16,384 optional);*
2) *Repertoire of 62 instructions, most of which provide for conditional program branches;*
3) *Average instruction execution time of 13 microseconds;*
4) *30-bit word length;*
5) *Optional operation with 15-bit half-words;*
6) *Internally stored program;*
7) *Programmed checking of data parity;*
8) *Parallel, ones' complement, subtractive arithmetic;*
9) *Single-address instructions with provision for address modification via seven index registers;*
10) *Internal 7-day real-time clock for initiating operations at desired times;*
11) *12 input and 12 output channels for rapid data exchanges with external equipment without program attention;*
12) *2 input and 2 output channels for intercomputer data transfer;*
13) *16-word wired auxiliary memory, for storage of critical instructions and constants, which provides facility for Automatic Recovery in event of program failure and for automatic initial loading of programs.*

2. GENERAL

The AN/USQ-20 Unit Computer emphasizes rapid communication with external devices and large, randomly accessible internal storage.

Single-address instructions are employed and have an average execution time of 13 microseconds. Instruction words are 30 bits; data words can be either 15 or 30 bits.

Internal storage of the Unit Computer consists of a 32,768-word ferrite core memory. Each word may be interpreted as a single 30-bit word, or as two 15-bit words individually addressed. Control, Arithmetic, and Input/Output sections of the computer each have access to the Storage section. A complete cycle for storage of a 30-bit word received from one of the other sections requires eight microseconds.

Arithmetic and logical operations are performed in the parallel binary mode. In most instances, the result of an operation appears in a 30-bit accumulator register. Arithmetic is ones' complement subtractive with a modulus (2^{30}-1).

Computer operation is controlled by a stored program capable of self-modification. Each program instruction contains a function code (6 bits), instruction operand designator (15 bits), and three execution modifiers (3 bits each). Execution modifiers provide for address incrementation, operand interpretation, and branch-point designation. The operand may be increased by the amount contained in any one of seven index registers. The operand specified by the execution address may be interpreted as a 30-bit quantity, or as a 15-bit half-word with or without sign extension. The next sequential program step may be skipped; it is under control of the content of the Accumulator or the Q-register.

Communication between the AN/USQ-20 Unit Computer and its associated external equipment is normally handled by a block transfer of data, with timing under control of the external device. Operating asynchronously with the main computer program, such transfers of data have independent access to storage.

A communication path is established by a sequence of request and response signals between external equipment and computer. Such signals may originate in either the computer or the external device. The main computer program is interrupted by external request signals and a communications channel is established. Once the link has been created, the computer returns to the main program sequence. Block transfer of input or output data then proceeds without program reference until completed.

A total of 14 input and 14 output channels is provided in the computer; each channel consists of 30 parallel lines. Two input and two output (special) channels are reserved for communication with other computers. The maximum possible transfer rate of input or output data over a given channel is greater than 30,000 words per second.

Output channels carry *External Function Words* as well as data words to external equipment. These specify the function desired of the external device. An External Function Word to a tape control unit, for example, may specify *Rewind Tape Unit 3.*

The computer (see Frontispiece) is housed in a single cabinet, 33 inches deep, 37 inches wide, 65 inches high. Thirteen trays, eight trays of logic modules and five trays of memory modules, are horizontally arrayed within the cabinet (Figure 1). Logic modules consist of encapsulated printed-circuit cards which plug into the trays. Maintenance test points are readily accessible at the front of the trays.

Computer cabinet doors, which are closed during normal operation, can be opened for maintenance. Inner surfaces of the doors contain maintenance control panels with register indicators, set and clear pushbuttons, and operating switches. A separate operating and maintenance console would be supplied for special applications, as in a multicomputer installation.

Primary power is provided to the computer from a 60-cycle input, 400-cycle output motor-alternator which, in addition to converting frequency, serves to isolate the computer from the main power source. Total power consumption is 2400 watts. For the installation planned for the AN/USQ-20 Unit Computer forced-air cooling is used; supplementary need for a heat exchanger is dictated by environment. Inter-equipment cabling enters the computer at the top of the cabinet and is run through a false floor.

The computer is designed and constructed to withstand severe shock and vibration. It may easily be installed aboard ship or in a trailer without special modification.

3. OPERATION

A simplified block diagram of the AN/USQ-20 Unit Computer appears in Figure 2. For explanatory purposes, the computer may be considered as comprised of four major sections: ***Input/Output***, ***Storage***, *Arithmetic,* and *Control*. Abbreviations on the diagram are explained as operation of the various sections is discussed.

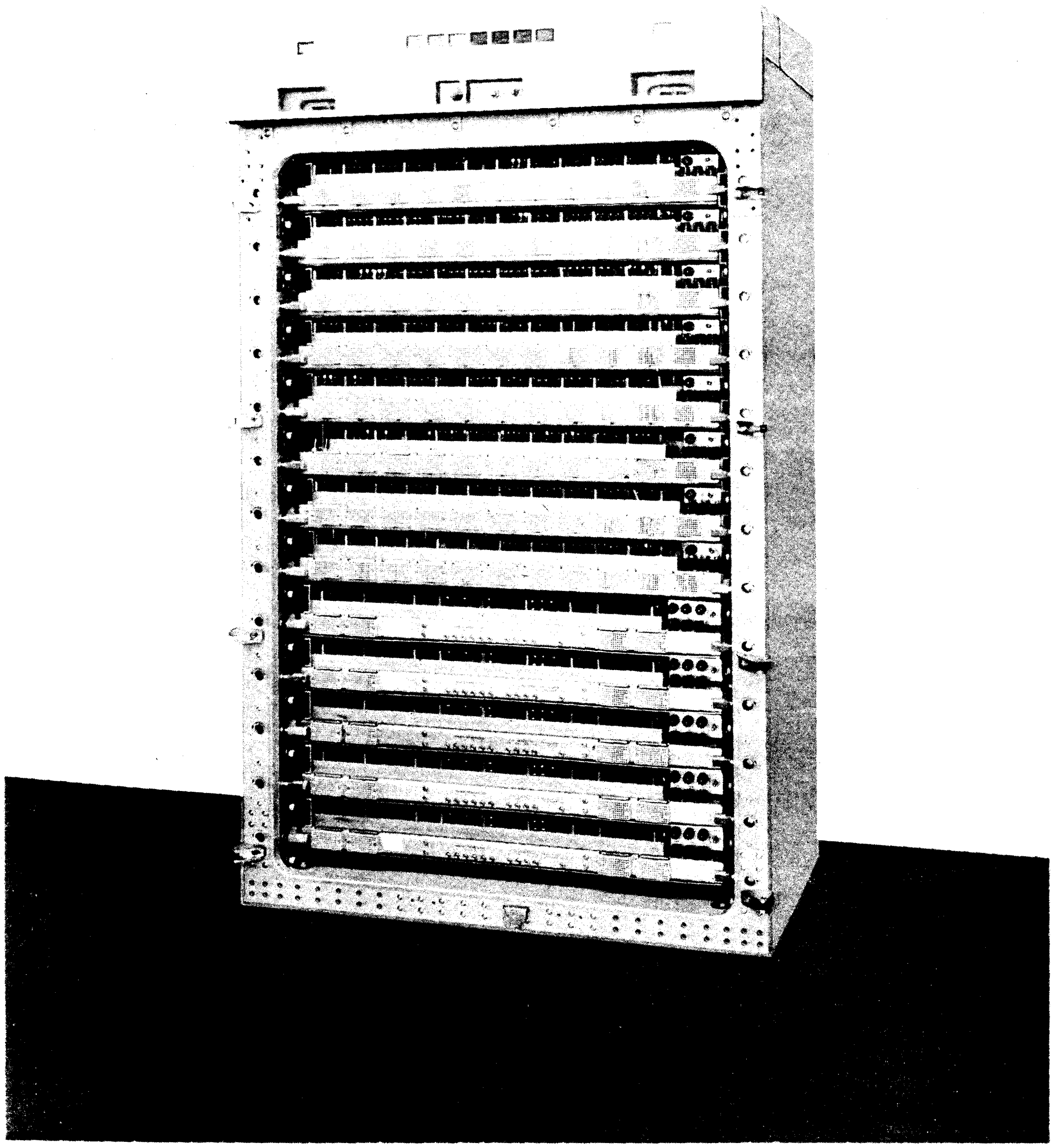


Figure 1. Computer Cabinet Interior

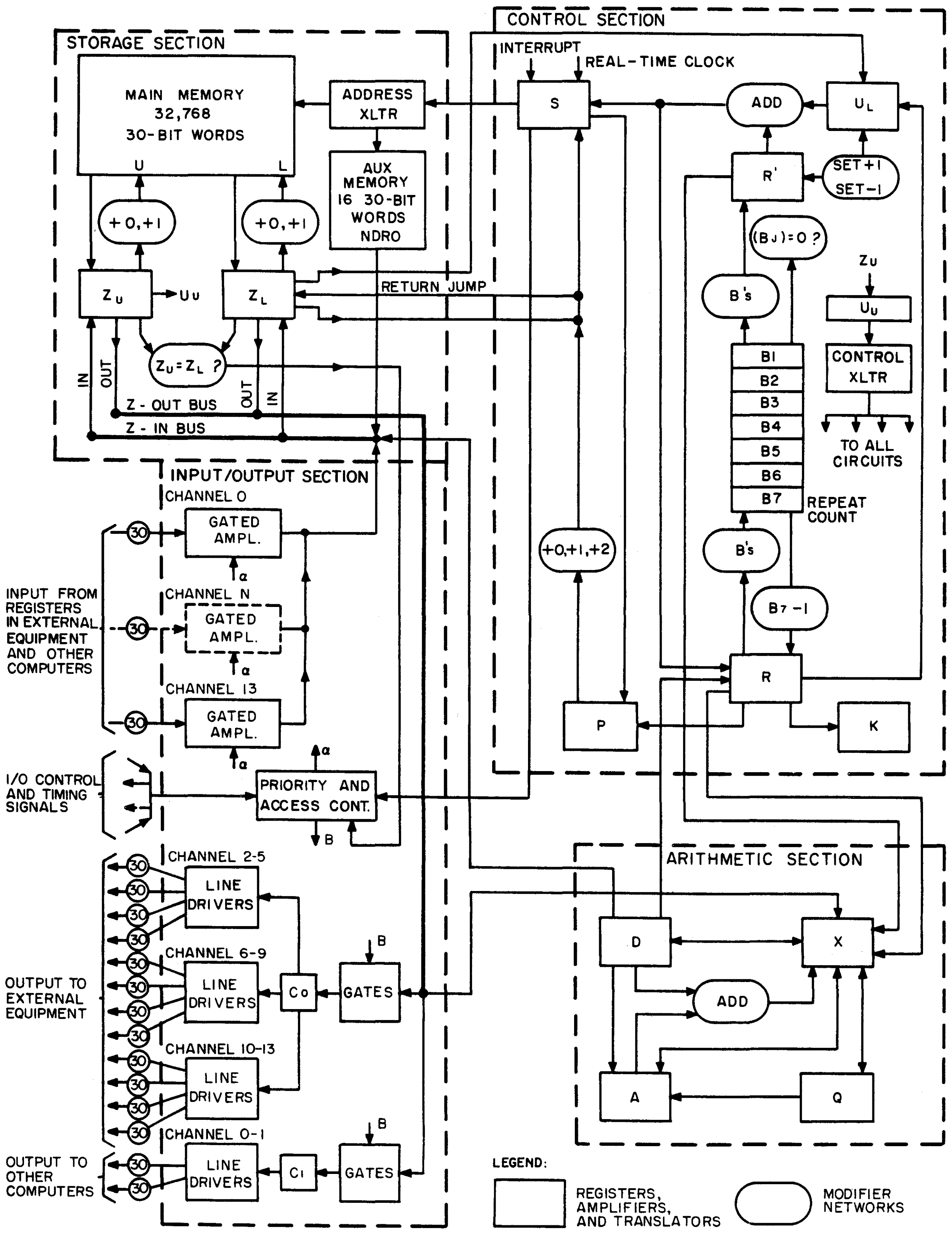


Figure 2. Simplified Block Diagram, AN/USQ-20 Unit Computer

INPUT/OUTPUT SECTION

The *Input/Output section* includes those data paths and control circuits used by the computer for communicating with external equipment. Main parts of the Input/Output section are 1) two output registers (C0 and C1) and their associated line drivers, 2) 14 sets of gated input amplifiers, and 3) priority and access control circuits.

Output Registers

The *C0-register* is used for transmissions to all external devices except other computers. As illustrated in Figure 2, C0 receives its input directly from the Storage section via gates controlled by the priority and access circuits. Three sets of 30 line drivers branch from the output of C0; each set drives four output channels. Gated registers located in the external devices determine which channel is active during any particular transmission.

The *C1-register* handles transmissions over the two special output channels — those used to transmit data to other computers. Operation of C1 is similar to that of C0: Words enter C1 from storage via gates controlled by the priority and access circuits and are transmitted over the active channel by a set of 30 line drivers.

Note that the output channels are numbered from 0 to 13. If two or more output transmissions are simultaneously requested, the channel with the highest number is granted priority; others follow in order.

Input Amplifiers

A set of *30 gated amplifiers* is provided for each of the 14 input channels. Gates are controlled by the priority and access circuits, with the channel having the highest number being given priority if two or more inputs are simultaneously requested. As in the case of the output channels, Channels 0 and 1 are used for intercomputer communication, which consequently receives the lowest priority.

This method of treating input data eliminates the need for input buffer registers and gives external equipment direct access to the computer's internal memory.

Priority and Access Circuits

Some functions of the *priority and access circuits* (namely, gating input and output transmissions and assigning priorities to the channels) have been previously described. In addition, these circuits accept and transmit the control and timing signals which must be exchanged between the computer and equipment with which it communicates.

The circuits also include a means of testing various channels to determine whether they are busy. This feature prevents the computer from attempting to communicate over a channel already in use.

Main memory addresses referenced during a particular input or output transfer are determined by a special I/O control word. One such word is assigned to each channel. It is sufficient at this point to note that a signal generated by the I/O control word is used by the priority and access circuits to deactivate the channel after the proper number of words has been transferred.

STORAGE SECTION

The *Storage section* consists of main memory, wired auxiliary memory, and associated address, transfer, and control circuits.

The main memory, constructed of modular arrays of ferrite cores, has a capacity of 32,768 words of 30 bits each, is coincident-current driven, and is addressed via the address translator. Content of the referenced address is read into the 30-bit Z-register. Because of optional use of 15-bit half-words, Z is split into two 15-bit sections termed Z-upper (Z_U) and Z-lower (Z_L).

The memory operates in the destructive read-out mode. Time required for the read/restore cycle is eight microseconds.

During the restore portion of the cycle, the content of Z_L or Z_U may be increased by one, as indicated by the +0, +1 modifier boxes. This provision allows for automatically increasing the I/O control words, with the result that addresses referenced during a block transfer of data are automatically advanced.

The comparator, $Z_U = Z_L$?, is used to detect coincidence between the two halves of the I/O control word. When coincidence occurs, a signal is generated to terminate the I/O transfer.

ARITHMETIC SECTION

The *Arithmetic section* is that part of the computer which performs numeric and logical calculations. Though greatly simplified, Figure 1 shows the important components of the Arithmetic section: the A-, D-, Q-, and X-registers and the add network.

The *A-register* (30 bits) may be thought of for programming purposes as a conventional accumulator. Because of the logic employed, however, the A-register is actually only the main rank of the accumulator; the *D-register* serves as a second rank. This configuration, while different from former and usual arrangements, permits the use of a much more reliable building-block circuit.

The Add operation is typical of the relationship between the A- and D-registers: The augend and addend are initially contained in A and D. As addition is performed, the sum is formed in parallel by the add network and placed in the *X-register*. From X, the sum is transmitted to A.

The *Q-register* (30 bits) is used principally during multiply and divide operations. The contents of both A and Q may be shifted left or right, individually or as one double-length 60-bit word.

CONTROL SECTION

The *Control section* consists of those registers and circuits necessary to procure; modify, and execute instructions of the program.

The *U-register* (30 bits) is the program-control register. It holds the instruction word during execution of an instruction. The function code and the various execution modifiers are translated from appropriate sections of the register. The lower-order 15 bits of the U-register have addition properties, modulus $2^{15}-1$. If an address modification is required before execution, contents of the appropriate *B-register* are added to contents of the lower-order 15 bits of the U-register before execution.

The *R-register* (15 bits) functions as a communication register for the B registers. All internal transmissions to or from B-registers pass through the R-register. It also holds the quantity used as increment during address modification. This register has counting provisions for increasing contents of the B-register.

The K-*register* (6 bits) functions as a shift counter for all arithmetic operations that involve shifts. The maximum shift count is 63. Multiply and divide operations are controlled by pre-setting the K-register to ZERO and counting operational steps.

The *S-register* (15 bits) holds the storage address during memory references. At the beginning of a storage access period, the address is transferred to the S-register. The contents of the S-register are then translated to activate the storage selection system.

4. CONSOLE CONTROL

Both the indicator/control panel in the doors (referred to as the *in-door console*) and the separate (optional) operating console include 1) indicator lamps that display a detailed report of the internal status of the computer, and 2) controls that allow varied manually governed operations. It is not necessary to monitor the consoles during normal operation.

REGISTERS

Each register is represented on a console by 1) a row of display lamps, each of which indicates the content of a corresponding register stage; 2) a row of SET buttons, each of which can be used to manually enter a *one* into the corresponding stage; and 3) a CLEAR button, which can be used to manually enter zeros into all stages of the register. Many of the registers are involved in the mechanics of executing instructions, and are not directly accessible to the program. (These registers are not discussed in this publication.)

SPECIAL MODES

Both the *in-door console* and the separate console are provided with manual controls that permit the following special modes of operation:

1) Execution of one program instruction of a sequence for each depression of a switch.
2) Execution of consecutive program instructions at a low rate governed by a console frequency control.
3) Execution of one master clock phase for each depression of a switch.
4) Execution of consecutive master clock phases at a low rate governed by a console frequency control.
5) Operation that is normal except that the computer does not stop when it executes a programmed *Stop* instruction. (Such operation is called abnormal high-speed operation.)

The consoles are also provided with a manual control that may be used to disable the real-time clock. This option enables the operator to suspend normal operation temporarily without affecting such operation when it is subsequently resumed. Such suspensions could include stopping the computer or operating temporarily in one of the special modes listed previously.

PROGRAMMED STOPS

The consoles include a set of six JUMP and STOP switches that can, in normal computer operation, govern the execution of manual-jump instructions. If *stop* conditions are satisfied, the computer stops and an indicator is lit to show the value of j in the manual-jump or manual return-jump instruction that stopped the computer. In abnormal high-speed operation, the indicator is lit but the computer does not stop.

MASTER CLEAR

The consoles include a spring-return key that is used to clear all registers and control designators to *zero*.

5. OTHER FEATURES

The AN/USQ-20 Unit Computer is especially well suited to real-time control, data processing, and data reduction. Most basic features of the computer are described in previous sections. A detailed account of how these features can be used is beyond the scope of this technical note. This section, however, suggests uses of certain computer features in a few areas.

Features considered here are: 1) automatic programming, 2) floating-point arithmetic, 3) the real-time clock, 4) masked comparison, 5) inclusion of operands in instruction words, 6) detection of special faults, 7) program branching, 8) search operations, 9) half-word and full-word logic, 10) external and internal program interrupts, and 11) 16-word wired auxiliary memory.

AUTOMATIC PROGRAMMING

Two phases of an extensive compiling system* have been completed and are currently in use.

*Additional references: 1) NTDS Technical Note No. 202, *Compiling System CS-1.*
2) *Compiling System CS-1 — Programmer's Reference Manual,* PX 1349.
2) *Phase III Basic Input Language*, PX 1478.

Early phases of the compiler allow for mnemonic expression of function codes and designators, permit relative addressing, and provide for automatic storage allocation. Instructions and addresses may be given alphanumeric names at the option of the programmer. There are,in addition,several operators which cause many instructions to be generated in the final program. A subroutine mechanism facilitates compilation of subroutines in the final program.

A table-handling routine is included to enable efficient table search and modification. Items in a table may be searched, extracted, or deleted.

Some important computational routines available are *square root*, *sine*, *cosine*, *tangent*, *arcsine*, and *arccosine*. Both direct computation and table look-up and interpolation routines are available for evaluating functions. The table look-up procedure can often be used to save considerable computation time, especially when adequate memory capacity is available.

A coordinate-conversion routine will convert polar coordinates to Cartesian coordinates and vice-versa. This routine is especially useful in control problems. Number conversion routines permit conversion from octal to decimal and from decimal to octal notation.

Compiling, in early stages of program checkout, can be done with special debugging aids which include post-mortem area and register dumps. A sampler routine provides for punching intermediate results during program operation; it does not require that the programmer provide for sampling while writing his program.

The third phase of the compiling system is in the late stages of debugging. This phase is a high-level language which allows programs to be written by means of powerful operators totally divorced from machine code. The operators cause many computer instructions to be generated in the final program. Some typical operators are *Procedure*, *Vary*, *Find*, *If*, *Then*, *Go To*, *Set*, *Resume*, *Return*, and *Switch*. Other operators make possible data definition into tables, sectors, items, fields, and subfields. Some operators are used to define variables. Procedures involving algebraic computation can be set up in their actual mathematical notation. In this unique approach to compiler development, all three phases will be compatible and available for developing other phases. The compiler is "open-ended"; additional functions can easily be added for specialized problems.

FLOATING-POINT ARITHMETIC PROGRAM PACKAGE

The floating-point format is based on a two-word information unit: one mantissa word and one characteristic word. The length of the mantissa word is 28 bits; the length of the characteristic word is 15 bits, including sign bit. It is a three-address system in which four B registers are used to designate operand and operation code. The average instruction time for a floating-point operation (*add, subtract, multiply,* or *divide*) is 500 microseconds.

REAL-TIME 7-DAY CLOCK

Among the features that suit the computer to real-time problems is the 7-day clock which contains an accurate record of time. The clock may be used to log the receipt times of a periodic real-time input. Each message and its receipt time may be recorded together. Another use of the clock is to initiate periodic programmed operations without requiring more than occasional attention of the main program. Since the clock recycles only once in 7 days, it is suitable for use where the computer is used on an around-the-clock basis.

MASKED COMPARISON

Masked Comparison is used to compare all or any part of a word with the contents of the accumulator. It also tests for *equality*, and *non-equality, greater than*, or *less than* conditions. In all cases, the original content of the Accumulator is left unchanged.

INCLUSION OF OPERANDS IN INSTRUCTION WORDS

The lower half (15 bits) of an instruction word is commonly used as an operand address. Where 15-bit operands are acceptable, the lower half of the instruction word may itself serve as an operand. This option of storing the operand as part of the instruction word is particularly advantageous for certain applications. It reduces computation by eliminating a memory reference and provides twice the storage capacity, if 15 bits are adequate for the precision required.

DETECTION OF SPECIAL FAULTS

Fault conditions may arise in the execution of a *divide* instruction: the divisor may be zero or the quotient may exceed 30 bits (including sign). Either *fault* condition is detected by programming the *divide* instruction with $j = 3$ (skip if Q is negative). When the instruction is executed, a fault produces a skip. The instruction that follows the one that is skipped contains a jump to a remedial subroutine. The instruction that immediately follows *divide* is programmed with $j = 1$ (unconditional skip) so that the remedial subroutine is avoided if no fault is detected. The computer never stops because of a divide fault.

When a *multiply* instruction is performed, the product is formed in AQ. If it does not exceed 30 bits (including sign bit), it is contained entirely in Q. If it exceeds 30 bits, it extends into A and is called a double-length product. A method of detecting a double-length product depends on these two facts:

1) During execution of a *multiply* instruction, both factors are arbitrarily represented as positive numbers (sign bit = 0); correction of the sign bit of the product occurs late in the program step.

2) Skip-condition evaluation takes place before sign correction.

The *multiply* instruction is programmed with j = 2 so that, if (Q_{29}) = *one* before sign correction, a skip is performed. If (Q_{29}) = *zero*, the test is inconclusive, and the next instruction tests for $(A) \neq 0$. The second test is conclusive: if $(A) \neq 0$, the product is double length; if $(A) = 0$, it is not. The additional instruction that contains the second test is skipped unless the first test is inconclusive. If the probable length of the product is known, the programmer can select a variation of this method (such as reversing the order of the two tests) to reduce the probability of having to execute an additional instruction.

PROGRAM BRANCHING

A convenient method of providing a branch in the program is to include a skip condition followed by a jump instruction. The skip is performed unless the branch condition obtains, so that the jump is not normally performed. If the branch condition obtains, however, the skip is not performed. The jump then diverts the program to the branch. This method (which may be used wherever optional use of an out-of-sequence address is desired) is advantageous because the skip is ordinarily part of some other instruction. Since the skip requires no separate instruction word, no execution time is spent on the branch condition except when that condition is satisfied. This arrangement conserves not only execution time, but memory capacity as well. Since the jump (like the skip) may be conditional, the arrangement simplifies inclusion of compound branch conditions.

Another feature that facilitates preparation of branched programs is the presence of return-jump instructions in the repertoire. When performance of a subroutine initiated by a main-program return-jump is completed, main-program operation is automatically resumed at the point at which it was interrupted.

SEARCH OPERATIONS

The computer can search internally stored files at very high speeds. The search identifier is entered into the Accumulator, and a mask that identifies the key (a *one* at every key bit position) is entered in Q. The search consists of a masked comparison that is performed repeatedly at successive memory addresses until a *find* (or other terminal condition) is detected.

HALF-WORD AND FULL-WORD LOGIC

All instructions that procure their operands from memory may select the upper 15 bits, the lower 15 bits, or the entire 30 bits as the operand. For some purposes, half-word accuracy is generally sufficient, and this feature can be used to reduce computation time and to double the effective memory capacity.

EXTERNAL AND INTERNAL PROGRAM INTERRUPTS

Provision is made for the interruption of running programs by events which occur asynchronously with the program.

External devices may, by placing a signal on one of 14 External Interrupt lines, interrupt the normal computer program in the event of a failure in data transmission or even in case of termination of some normal mode of operation (in case of a tape system, at end of rewind). Appropriate action is taken by the computer's interrupt program and the normal program is re-entered at the point at which it was left.

Moreover, interrupts are generated by the I/O section of the computer whenever a buffer, which has been initiated previously with a monitor imposed upon it, terminates at the end of the transfer of a given block of data. The interrupt program takes cognizance of the buffer termination, and the main program is resumed.

16-WORD WIRED AUXILIARY MEMORY

In addition to the large main memory, a 16-word auxiliary memory is also provided. It is a wired memory and operates in the nondestructive read-out mode. The auxiliary memory is used to contain important instructions or constants. For example, a program-load routine may be stored there to facilitate rapid changes in the main program and automatic program recovery.

APPENDIX A

REPERTOIRE OF INSTRUCTIONS AND PROGRAM TIMING

APPENDIX A

REPERTOIRE OF INSTRUCTIONS AND PROGRAM TIMING

1. INTRODUCTION

This portion of the technical note presents the instruction repertoire for the AN/USQ-20 Unit Computer. Details presented are limited to the needs of the programmer and list only symbols, registers, terms, and instruction characteristics pertinent to programming the computer.

As mentioned previously, the AN/USQ-20 Unit Computer is a self-modifying, one-address computer. Although this means that one reference or address is provided for the execution of an instruction, this reference can be modified automatically during a programmed sequence. The references are modified by using the B (index) registers one through seven, which contain any previously stored constants. To modify the address, the content of a selected B-register is added to the Operand Designator, **y**.

A programmed address is coded using octal notation with each octal digit denoting three binary digits. The instructions are read sequentially from Magnetic Core Storage except after Jump or Skip instructions.

A. *SYMBOL CONVENTIONS* - The following symbols are used throughout the descriptive material on instructions:

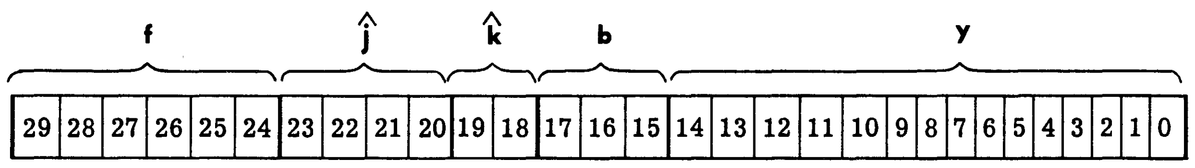
α	=	a register (A, Q, B^n), a memory location Y, or a constant.
(α)	=	content of α.
$(\alpha)_i$	=	initial content of α.
$(\alpha)_f$	=	final content of α.
α_n.	=	the n^{th} bit of α.
$(\alpha)_n$	=	the n^{th} bit of the content of α.
f	=	Function Code Designator ($i_{29}, \ldots, i_{24}$)*.
j	=	Branch Condition Designator ($i_{23}, \ldots, i_{21}$)*.
$\hat{\mathbf{j}}$	=	Input/Output Channel Designator ($i_{23}, \ldots, i_{20}$)*.
k	=	Operand Interpretation Designator ($i_{20}, \ldots, i_{18}$)*.
$\hat{\mathbf{k}}$	=	Operand Interpretation Designator ($i_{19}, \ldots, i_{18}$)*.
b	=	Index Designator (i_{17}, i_{16}, i_{15})*.
y	=	Operand Designator ($i_{14}, \ldots, i_0$)*.
Y	=	the Operand (regardless of source).
Y	=	**y** + (B^b). 1) The operand or address of the operand for the *Read* portion of an instruction or 2) The destination address for the *Store* portion of an instruction.
(Y)	=	content of memory address Y.
L(Y)(Q)	=	bit-by-bit multiplication, logical multiply of $\mathbf{Y}_n$, and $(Q)_n$.
A	=	A-register or accumulator (30-bit arithmetic register).
B	=	seven B-registers (15 bits each). B-registers are address-modifying registers generally used for indexing loops in a program; in addition, B^7 serves as a *repeat* counter. (The address modification does not alter the instructions as stored in memory.) A **b** or **j** designator specifies the B-register used.
Q	=	Q-register (30-bit arithmetic register).
U	=	U-register (30 bits). The U-register holds the instruction word during execution of an operation. If address modification is required before execution, the appropriate B-register content is added to the lower-order 15 bits of the U-register before execution.

* i_n is the n^{th} bit position in an instruction.

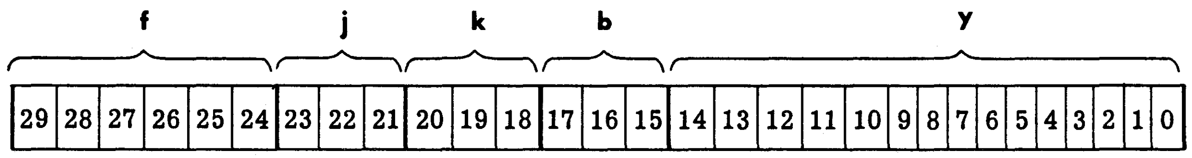
P = P-register (15 bits). The P-register is the Program Address Register. This register holds the address of the current instruction throughout the program except for Jump instructions where the P-register is cleared and the new program address is entered.

C = the 14D input/output channels (30 lines each). Channels consist of transmission lines, therefore they *cannot* be considered registers. The designator $\hat{j}$ specifies (in octal) the channel used.

Figure A-1 illustrates bit configuration of instruction designators in two forms. Form I pertains to input/output instructions; Form II pertains to all other instructions.



Form I - Input/Output Instructions



Form II - All Other Instructions

Note: $\hat{j}$ = C^n input/output channel

Figure A-1. Bit Allocation of Instruction Designators

Table A-1 is a list of the computer's entire repertoire of instructions; each instruction is listed by its function code number and name.

Table A-2 indicates the time, in microseconds, required to execute each instruction.

TABLE A-1. INSTRUCTION REPERTOIRE - AN/USQ-20 UNIT COMPUTER

CODE	FUNCTION NAME	CODE	FUNCTION NAME
00	(Fault Interrupt)	40	ENTER LOGICAL PRODUCT
01	RIGHT SHIFT Q	41	ADD LOGICAL PRODUCT
02	RIGHT SHIFT A	42	SUBTRACT LOGICAL PRODUCT
03	RIGHT SHIFT AQ	43	COMPARE MASKED
04	COMPARE	44	REPLACE LOGICAL PRODUCT
05	LEFT SHIFT Q	45	REPLACE A + LOGICAL PRODUCT
06	LEFT SHIFT A	46	REPLACE A - LOGICAL PRODUCT
07	LEFT SHIFT AQ	47	STORE LOGICAL PRODUCT
10	ENTER Q	50	SELECTIVE SET
11	ENTER A	51	SELECTIVE COMPLEMENT
12	ENTER B^n	52	SELECTIVE CLEAR
13	EXTERNAL FUNCTION ON C^n	53	SELECTIVE SUBSTITUTE
14	STORE Q	54	REPLACE SELECTIVE SET
15	STORE A	55	REPLACE SELECTIVE COMPLEMENT
16	STORE B^n	56	REPLACE SELECTIVE CLEAR
17	STORE C^n	57	REPLACE SELECTIVE SUBSTITUTE
20	ADD A	60	JUMP (Arithmetic)
21	SUBTRACT A	61	JUMP (Manual)
22	MULTIPLY	62	JUMP ON C^n ACTIVE INPUT BUFFER
23	DIVIDE	63	JUMP ON C^n ACTIVE OUTPUT BUFFER
24	REPLACE A + Y	64	RETURN JUMP (Arithmetic)
25	REPLACE A - Y	65	RETURN JUMP (Manual)
26	ADD Q	66	TERMINATE C^n INPUT BUFFER
27	SUBTRACT Q	67	TERMINATE C^n OUTPUT BUFFER
30	ENTER Y + Q	70	REPEAT
31	ENTER Y - Q	71	B SKIP ON B^n
32	STORE A + Q	72	B JUMP ON B^n
33	STORE A - Q	73	INPUT BUFFER ON C^n (without Monitor mode)
34	REPLACE Y + Q	74	OUTPUT BUFFER ON C^n (without Monitor mode)
35	REPLACE Y - Q	75	INPUT BUFFER ON C^n (with Monitor mode)
36	REPLACE Y + 1	76	OUTPUT BUFFER ON C^n (with Monitor mode)
37	REPLACE Y - 1	77	(Fault Interrupt)

TABLE A-2. INSTRUCTION EXECUTION TIMES

	j=0,1 NORMAL			j=0 REPEAT		
f	k=0,4	k=7	k≠0,4,7	k=0,4	k=7	k≠0,4,7
01	9.6/12.8	11.2/14.4	16			
02	9.6/12.8	11.2/14.4	16			
03	11.2/16	11.2/16	16/20.8			
04	12.8	11.2	16	9.6	8.0	11.2
05	9.6/12.8	9.6/12.8	16			R
06	9.6/12.8	9.6/12.8	16			R
07	11.2/16	11.2/16	16/20.8			R
10	11.2	9.6	16	8.0	6.4	9.6 R
11	11.2	9.6	16	8.0	6.4	9.6
12	8.0	9.6	16	4.6	6.4	9.6
13	12.8	-	24			
S 14	12.8	-	16	6.4	-	9.6
S 15	12.8	-	16	6.4	-	9.6
S 16	12.8	-	16	6.4	-	9.6
S 17	-	-	16			
20	11.2	9.6	16	8.0	6.4	9.6
21	11.2	9.6	16	8.0	6.4	9.6
22	35.2 - 112					
23	112					
R 24	-	-	24	-	-	16
R 25	-	-	24	-	-	16
26	12.8	11.2	16	9.6	8.0	11.2
27	12.8	11.2	16	9.6	8.0	11.2

	j=0,1 NORMAL			j=0 REPEAT		
f	k=0,4	k=7	k≠0,4,7	k=0,4	k=7	k≠0,4,7
30	11.2	9.6	16	8	6.4	9.6
31	11.2	9.6	16	8	6.4	9.6
S 32	12.8	-	16	6.4	-	9.6
S 33	12.8	-	16	6.4	-	9.6
R 34	-	-	24	-	-	16
R 35	-	-	24	-	-	16
R 36	-	-	24	-	-	16
R 37	-	-	24	-	-	16
40	11.2	9.6	16	8	6.4	9.6
41	11.2	9.6	16	8	6.4	9.6
42	11.2	9.6	16	8	6.4	9.6
43	11.2	9.6	16	8	6.4	9.6
R 44	-	-	24	-	-	16
R 45	-	-	24	-	-	16
R 46	-	-	24	-	-	16
S 47	12.8	-	16	6.4	-	9.6
50	12.8	11.2	16	9.6	8.0	11.2
51	12.8	11.2	16	9.6	8.0	11.2
52	12.8	11.2	16	9.6	8.0	11.2
53	12.8	11.2	16	9.6	8.0	11.2
R 54	-	-	24	-	-	16
R 55	-	-	24	-	-	16
R 56	-	-	24	-	-	16
R 57	-	-	24	-	-	16

	j=0 NORMAL		
f	k=0,4	k=7	k≠0,4,7
60	8	9.6	16
61	8	9.6	16
62	8		16
63	8		16
64	12.8/19.2	11.2/17.6	16/24
65	12.8/19.2	11.2/17.6	16/24
66	8	-	16
67	8	-	16
70	8	9.6	16
71	9.6	11.2	16
72	8	9.6	16
73	16	-	24
74	16	-	24
75	16	-	24
76	16	-	24
77	-	-	-

S - STORE

R - REPLACE

Note:

All times are in microseconds

B. *FUNCTION CODE DESIGNATOR* - f

The f designator (6 bits) appears in bit-positions 29 through 24 of the U-register, or an instruction, designating the function to be performed by that instruction. All values of f other than 00 and 77 are defined in the instruction list. The two codes 00 and 77 are *fault conditions* which, if executed, will cause a *fault interrupt.* This results in a jump to address 00014, the Fault Entrance Register or address 00014 of *wired* memory depending on the Automatic Recovery Switch setting (see page A-10).

C. *BRANCH CONDITION DESIGNATOR* - j

The j designator (3 bits) appears in bit-positions 23, 22, and 21 of the U-register, or an instruction; it is used in a majority of the instructions (see Figure A-1, Form II). There are three primary categories of use: 1) for Jump and Skip determination, 2) for B-register specification, and 3) for repeat status interpretation. Appropriate interpretations of the j designator are listed either below or under the descriptions of the individual instructions.

For those instructions in which the j designator has no special interpretation, it specifies the condition under which the next sequential instruction in the program will be skipped. This provides for branching from a sequence without executing a Jump instruction, as would normally occur if a Skip condition were not satisfied.

Skip of the next sequential instruction is determined by the following rules in all instructions except 04, 12, 13, 16, 17, 26, 27, 60 through 67, and 70 through 76.

j = 0:	Do not skip the next instruction.
j = 1:	Skip the next instruction.
j = 2:	Skip the next instruction if (Q) is positive.
j = 3:	Skip the next instruction if (Q) is negative.
j = 4:	Skip the next instruction if (A) is zero.*
j = 5:	Skip the next instruction if (A) is nonzero.
j = 6:	Skip the next instruction if (A) is positive.
j = 7:	Skip the next instruction if (A) is negative.

When the branch (Skip or Jump) condition involves the sign of the quantity in A or Q, the evaluation examines the sign bit of these quantities; hence, a positive zero (all *zeros*) is considered a positive quantity, and a negative zero (all *ones*) is considered a negative quantity.

* Positive zero

D. *INPUT/OUTPUT CHANNEL DESIGNATOR* - $\hat{j}$

The $\hat{j}$ designator (4 bits) appears in bit-positions 23, 22, 21, and 20 of the U-register, or an input/output instruction, specifying the C-channel for the instruction (see Figure A-1, Form I). Bit 23 assumes a value of eight, bit 22 a value of four, bit 21 a value of two, and bit 20 a value of one; thus the $\hat{j}$ designator provides accessibility to the 14 (decimal) input/output channels numbered $0\text{-}15_8$.

Instructions 13, 17, 62, 63, 66, 67, 73, 74, 75, and 76 use the $\hat{j}$ designator configuration.

E. *OPERAND INTERPRETATION DESIGNATOR* - k or $\hat{k}$

The k designator (3 bits) [or $\hat{k}$ designator (2 bits)] appears in bit-positions 20, 19, and 18 of the U-register, or an instruction; a $\hat{k}$ designator appears only in bit-positions 19 and 18, since bit 20 is a portion of the $\hat{j}$ designator. (See Figure A-1, Forms I and II.) Instructions 13, 17, 62, and 73 through 76 use the $\hat{k}$ designator configuration since they perform input/output activities and require a $\hat{j}$ designator for channel specification.

The k and $\hat{k}$ designators control operand interpretation. Those instructions which *read* an operand but do not replace it after the arithmetic is performed are designated *Read* instructions. Those instructions which do not *read* an operand but *store* one are designated *Store* instructions. Instructions which both *read* and *store* operands are classified as *Replace* instructions.

The various values of k or $\hat{k}$ affect the operand in the following list except where otherwise noted under individual instruction descriptions.

1) *Read* instructions (01 through 13, 20 through 23, 26, 27, 30, 31, 40 through 43, 50 through 53, and 60 through 76):

k or $\hat{k}$ = 0: Y_u = 0's; Y_L = Y.

k or $\hat{k}$ = 1: Y_u = 0's; $Y_L = (Y)_L$.

k or $\hat{k}$ = 2: Y_u = 0's; $Y_L = (Y)_u$.

k or $\hat{k}$ = 3: Y = Y.

k = 4: Y_u = same bits as Y_{14}; Y_L = Y.

k = 5: Y_u = same bits as Y_{14}; $Y_L = (Y)_L$.

k = 6: Y_u = same bits as Y_{29}; $Y_L = (Y)_u$.

k = 7: Y = (A).

For instructions 23, 52, and 53, $\mathbf{k} = 7$ is not used.

For instruction 13, only $\hat{\mathbf{k}} = 3$ is permitted.

For instructions 73 through 76, $\hat{\mathbf{k}} = 2$ is not used.

2) *Store* instructions (14 through 16, 17, 32, 33, and 47):

$\mathbf{k} = 0$:	Store (A or B^j) in Q^*.
$\mathbf{k} = 1$:	Store (A_L, Q_L, or B^j) in Y_L, leaving $(Y)_u$ undisturbed.
$\mathbf{k} = 2$:	Store (A_L, Q_L, or B^j) in Y_u, leaving $(Y)_L$ undisturbed.
$\mathbf{k}$ or $\hat{\mathbf{k}} = 3$:	Store (A, Q, C^j, or B^j) in Y.
$\mathbf{k} = 4$:	Store (Q or B^j) in A^{**}.
$\mathbf{k} = 5$:	Store complement of (A_L, Q_L, or B^j) in Y_L, leaving $(Y)_u$ undisturbed.
$\mathbf{k} = 6$:	Store complement of (A_L, Q_L, or B^j) in Y_u, leaving $(Y)_L$ undisturbed.
$\mathbf{k} = 7$:	Store complement of (A, Q, or B^j) in Y. (Storing the complement of B^j is the same complement as for a 30-bit register.)

For instruction 17, only $\hat{\mathbf{k}} = 3$ is permitted.

3) *Replace* instructions (24, 25, 34 through 37, 44 through 46, and 54 through 57):

$\mathbf{k} = 0$:	Not used.
$\mathbf{k} = 1$:	*Read* portion - $\mathbf{Y}_u$ = 0's; $\mathbf{Y}_L = (Y)_L$. *Store* portion - stores (A_L, Q_L, or B^j) in Y_L, leaving $(Y)_u$ undisturbed.
$\mathbf{k} = 2$:	*Read* portion - $\mathbf{Y}_u$ = 0's; $\mathbf{Y}_L = (Y)_u$. *Store* portion - stores (A_L, Q_L, or B^j) in Y_u, leaving $(Y)_u$ undisturbed.
$\mathbf{k} = 3$:	*Read* portion - $\mathbf{Y}$ = Y. *Store* portion - stores (A, Q, or B^j) in Y.
$\mathbf{k} = 4$:	Not used.
$\mathbf{k} = 5$:	*Read* portion - $\mathbf{Y}_u$ = same bits at Y_{14}; $Y_L = (Y)_L$. *Store* portion - stores (A_L, Q_L, or B^j) in Y_L, leaving $(Y)_u$ undisturbed.

* A 14000 00000 instruction complements (Q).

** A 15040 00000 instruction complements (A).

k = 6: *Read* portion - Y_u = same bits as Y_{29}; $Y_L = Y_u$.

Store portion - stores (A_L, Q_L, or B^j) in Y_u, leaving $(Y)_L$ undisturbed.

k = 7: Not used.

The *Repeat* instruction requires special interpretation when followed by a *Replace* instruction. See details on page A-22, Instruction No. 70, *REPEAT*.

F. *INDEX DESIGNATOR* - b

The b designator (3 bits) appears in bit-positions 17, 16, and 15 of the U-register, or an instruction (see Figure A-1), specifying which of the B-registers, if any, will be used to modify the Operand Designator, y, to form $Y = y + (B^b)$. This operation employs an additive accumulator; hence, a quantity consisting of all *zero* cannot result unless the bits of both the Operand Designator, y, and (B^b) are all *zeros*.

Effect of the various values of b, the Index Designator, is summarized:

b = 0:	Do not modify y.
b = 1:	Add (B^1) to y (modulo 2^{15}-1).
b = 2:	Add (B^2) to y (modulo 2^{15}-1).
b = 3:	Add (B^3) to y (modulo 2^{15}-1).
b = 4:	Add (B^4) to y (modulo 2^{15}-1).
b = 5:	Add (B^5) to y (modulo 2^{15}-1).
b = 6:	Add (B^6) to y (modulo 2^{15}-1).
b = 7:	Add (B^7) to y (modulo 2^{15}-1).

G. *OPERAND DESIGNATOR* - y

The y designator (15 bits) appears in bit-positions 14 through 0 of an instruction (see Figure A-1). The operand or address of the operand, Y, is relative to y since $Y = y + (B^b)$.

H. *MAGNETIC CORE MEMORY ASSIGNMENT*

The main Magnetic Core memory consists of 32,768 addressable storage locations. Seventy-three of these locations are special-purpose and provide eight distinct functions:

1) The starting address from MASTER CLEAR
2) The Fault Entrance Register
3) The Real-Time Clock Register
4) External Interrupt Entrance Register for each channel
5) Internal Interrupt Entrance Register for each *input* channel
6) Internal Interrupt Entrance Register for each *output* channel
7) Input Buffer Control Register for each *input* channel
8) Output Buffer Control Register for each *output* channel.

Each of the other memory locations are used for:

1) Instruction word storage
2) Data storage.

I. *WIRED MEMORY* - The AN/USQ-20 Unit Computer contains 16D words of semipermanent wired storage. Programming this memory area requires a process of *wiring-in* the desired instructions. The semipermanent feature of these storage locations prevents accidental destruction of program instructions contained therein since entries cannot be made via main memory.

An Input Bootstrap routine occupies this memory, and its execution is controlled by the Automatic Recovery Switch.

J. *AUTOMATIC RECOVERY* - In the event of a *fault* condition (encountering either a 00 or 77 function code), the Automatic Recovery Switch directs computer activity. This switch has three positions: 1) DOWN, 2) NEUTRAL, and 3) UP. Action resulting from these positions is:

1) DOWN position - This causes manual execution of the Bootstrap routine. Computer action begins at address 0 of Wired Memory and executes the Bootstrap routine when this switch is depressed. (A MASTER CLEAR should precede this operation.)

2) NEUTRAL position - This causes an Interrupt to address 00014 of Main Memory on a fault condition. Action continues as programmed.

3) UP position - This causes an Interrupt to address 14 of Wired Memory on a fault condition. This results in automatic execution of the Bootstrap routine.

K. *BUFFER MODES* - The AN/USQ-20 Unit Computer provides two modes of buffering: 1) with monitor and 2) without monitor.

Buffering with monitor transfers words sequentially, starting at a given initial address through a given terminal address, on the specified input or output channel. The computer continues execution of program instructions during the buffer process. Completion of the buffering process causes an Internal Monitor Interrupt to the Internal Interrupt Entrance Register assigned to the input or output channel. (See subsection H, *MAGNETIC CORE MEMORY ASSIGNMENT*.) This register should contain a RETURN-JUMP instruction*. (See Instructions 75 and 76.)

Buffering without the monitor transfers words sequentially, starting at a given initial address through a given terminal address, on a specified input or output channel. The computer continues execution of program instructions during the buffer process. No monitor interrupt will occur. (See Instructions 73 and 74.)

2. LIST OF INSTRUCTIONS

This section lists the repertoire of instructions used with the AN/USQ-20 Unit Computer. Common usage of these instructions is also included; no attempt is made to indicate more sophisticated use.

01 *RIGHT SHIFT Q*

This instruction shifts (Q) to the right Y bit positions. The higher-order bits are replaced with the original sign bit as the word is shifted. Only the lower-order six bits of Y are recognized for this instruction. The higher-order 24 bits are ignored.

Example of right shift in Q: Y = 2

Content of Q		Content of Q	
$(Q)_i$ (positive) =	0 1 0 1	$(Q)_i$ (negative) =	1 0 1 0
First shift	0 0 1 0	First shift	1 1 0 1
Second shift	0 0 0 1	Second shift	1 1 1 0

02 *RIGHT SHIFT A*

This instruction shifts (A) to the right Y bit positions. The higher-order bits are replaced with the original sign bit as the word is shifted. Only the lower-order six bits of

* Suggested instruction for the Internal Interrupt Register is:
650nn nnnnn - Exit to an Interrupt subroutine for remedial action. This subroutine ends with a 601nn instruction which clears the Interrupt mode, then returns control to the main routine.

Y are recognized for this instruction. The higher-order 24 bits are ignored. The overall operation is analogous to the example given in the foregoing instruction.

03 *RIGHT SHIFT AQ*

This instruction shifts (A) and (Q) as one 60-bit register. The shift is to the right Y bit positions with the lower-order bits of A shifting into the higher-order bit positions of Q. The higher-order bits of A are replaced with the original sign bit as the word is shifted. Only the lower-order six bits of Y are recognized for this instruction. The higher-order 24 bits are ignored.

Example of right shift in AQ: Y = 2

Content of AQ		Content of AQ	
$(AQ)_i$ (positive) =	0 1 0 1 0 0 1 1	$(AQ)_i$ (negative) =	1 0 0 0 1 0 1 0
First shift	0 0 1 0 1 0 0 1	First shift	1 1 0 0 0 1 0 1
Second shift	0 0 0 1 0 1 0 0	Second shift	1 1 1 0 0 0 1 0

04 *COMPARE*

This instruction compares the signed value of Y with the signed value of (A) and/or (Q). It does not alter either (A) or (Q). The Branch Condition Designator, j, is interpreted in a special way for this instruction as listed below:

j = 0:	Do not skip the next instruction.
j = 1:	Skip the next instruction.
j = 2:	Skip the next instruction if Y is less than, or equal to, (Q).
j = 3:	Skip the next instruction if Y is greater than (Q).
j = 4:	Skip the next instruction if (Q) is greater than, or equal to Y, and Y is greater than (A).
j = 5:	Skip the next instruction if Y is greater than (Q) or if Y is less than, or equal to, (A).
j = 6:	Skip the next instruction if Y is less than, or equal to, (A).
j = 7:	Skip the next instruction if Y is greater than (A).

05 *LEFT SHIFT Q*

This instruction shifts (Q) circularly to the left Y bit positions*. The lower-order bits

* Maximum shift count permitted is 59D places.

are replaced with the higher-order bits as the word is shifted. Only the lower-order six bits of Y are recognized for this instruction. The higher-order 24 bits are ignored.

Example of left circular shift in Q: (Y) = 2

Content of Q		Content of Q	
$(Q)_i$ (positive) =	0 0 1 1	$(Q)_i$ (negative) =	1 1 0 0
First shift	0 1 1 0	First shift	1 0 0 1
Second shift	1 1 0 0	Second shift	0 0 1 1

06 *LEFT SHIFT A*

This instruction shifts (A) circularly to the left Y bit positions.* The lower-order bits are replaced with the higher-order bits as the word is shifted. Only the lower-order six bits of Y are recognized for this instruction. The higher-order 24 bits are ignored. The over-all operation is analogous to the example given in the foregoing instruction.

07 *LEFT SHIFT AQ*

This instruction shifts (A) and (Q) as one 60-bit register. The shift is circular to the left Y bit positions.* The lower-order bits of A are replaced with the higher-order bits of Q and the lower-order bits of Q are replaced with the higher-order bits of A. Only the lower-order six bits of Y are recognized by this instruction. The higher-order 24 bits are ignored.

Example of left circular shift in AQ: Y = 2

Content of AQ		Content of AQ	
$(AQ)_i$ (positive) =	0 1 0 1 0 0 1 1	$(AQ)_i$ (negative) =	1 0 0 0 1 0 1 1
First shift	1 0 1 0 0 1 1 0	First shift	0 0 0 1 0 1 1 1
Second shift	0 1 0 0 1 1 0 1	Second shift	0 0 1 0 1 1 1 0

10 *ENTER Q*

Clear the Q-register. Then transmit Y to Q.

11 *ENTER A*

Clear A. Then transmit Y to A.

12 *ENTER* B^n

Clear B-register j. Then transmit the lower-order 15 bits of Y to B-register j. The higher-order 15 bits of Y are ignored in this instruction. The Branch Condition Desig-

* Maximum shift count permitted is 59D places.

nator, $\mathbf{j}$, is used to specify the selected B-register for this instruction and is not available for its normal function.

13 *EXTERNAL FUNCTION ON C^n*

$\hat{j}$ = 0 or 1. Interrogate the two bits connected to the input-active designator (flip-flops) on an interconnected computer. If the interconnected computer's input buffer is active, skip the next instruction. If the interconnected computer's input buffer is not active, execute the next instruction. There are no External Function lines on C^0 or C^1. $\hat{k}$ = 3 is required for timing. When $\hat{j} \neq 0$ or 1, transmit $\mathbf{Y}$, the External Function, over the channel specified by $\hat{j}$. Only $\hat{k}$ = 3 is permitted.

14 *STORE Q*

Store (Q) at storage address Y as directed by the Operand Interpretation Designator, $\mathbf{k}$. If $\mathbf{k}$ = 0, complement (Q). If $\mathbf{k}$ = 4, store in A.

15 *STORE A*

Store (A) at storage address Y as directed by the Operand Interpretation Designator, $\mathbf{k}$. If $\mathbf{k}$ = 4, complement (A). If $\mathbf{k}$ = 0, store in Q.

16 *STORE B^n*

Store a 30-bit quantity whose lower-order 15 bits correspond to the content of B-register $\mathbf{j}$ and whose higher-order 15 bits are *zero* at storage address Y as directed by the Operand Interpretation Designator, $\mathbf{k}$. The Branch Condition Designator, $\mathbf{j}$, is used to specify the selected B-register for this instruction and is not available for its normal function.

17 *STORE C^n*

Store the content of the C-channel specified by $\hat{j}$ at storage address Y. An Input Acknowledge signal is then sent on the C-channel. Only $\hat{k}$ = 3 is permitted.

20 *ADD A*

Add $\mathbf{Y}$ to the previous content of the Accumulator.

21 *SUBTRACT A*

Subtract $\mathbf{Y}$ from the previous content of the Accumulator.

22 *MULTIPLY*

Multiply (Q) times $\mathbf{Y}$ leaving the double-length product in AQ. If the factors are considered as integers, the product is an integer in AQ.

The Branch Condition Designator, j, is interpreted prior to end correction permitting sensing of a product with $(A)_f = 0$. If j equal 4, a skip of the next instruction is made when $(A)_f = 0$. When $(A)_f \neq +0$, a double-length product has been formed with significant bit(s) in the Accumulator; however, if a Skip does occur for j = 4, the Multiply instruction can be re-executed with the same operand and with j = 2 or 3 to determine if Q_{29} contains a significant bit (a *one*) of the product.

In this instruction, **k** = 7 should not be used.

23 *DIVIDE*

Divide (AQ) by **Y** leaving the quotient in the Q-register and the remainder in the A register. The remainder bears the same sign as the quotient. In this instruction, **k** = 7 should not be used.

NOTE:

If a DIVIDE FAULT condition exists, no Maintenance Console indication is given; however, by coding each Divide instruction with j = 3, a program test for the DIVIDE FAULT is automatic. With this selection of j, a Skip of the next instruction occurs if a DIVIDE FAULT exists. The Skip should be made to a Jump instruction which provides a remedial means of noting or correcting the error. Therefore, the instruction which follows the Divide instruction should have its j = 1 in order to preclude the Jump instruction whenever the "Divide Sequence" culminates in a correct answer.
A DIVIDE FAULT can also be detected if the Divide instruction is executed with j = 2. In this case, a correct answer is indicated when a Skip occurs.

24 *REPLACE A + Y*

Add (Y) to the previous content of A. Store (A) at storage address Y.

25 *REPLACE A - Y*

Subtract (Y) from the previous content of A. Then store (A) at storage address Y.

26 *ADD Q*

Interchange (A) and (Q). Then add **Y** to (A). Interchange (A) and (Q). The content of A is undisturbed by this instruction. The Branch Condition Designator, j, has special meaning in this instruction as in instruction 27.

27 *SUBTRACT Q*

Interchange (A) and (Q). Then subtract **Y** from (A). Interchange (A) and (Q). The content of A is undisturbed by this instruction. The Branch Condition Designator, **j**, has special meaning in this instruction as listed below.

NOTE:

In instructions 26 and 27 the Branch Condition Designator, **j**, *has the following meaning:*

j = *0:*	*Do not skip the next instruction.*
j = *1:*	*Skip the next instruction.*
j = *2:*	*Skip the next instruction if (A) is positive.*
j = *3:*	*Skip the next instruction if (A) is negative.*
j = *4:*	*Skip the next instruction if (Q) is zero.*
j = *5:*	*Skip the next instruction if (Q) is nonzero.*
j = *6:*	*Skip the next instruction if (Q) is positive.*
j = *7:*	*Skip the next instruction if (Q) is negative.*

30 *ENTER Y + Q*

Clear A. Then transmit (Q) to A. Then add **Y** to (A).

31 *ENTER Y - Q*

Clear A. Then transmit (Q) to A. Then subtract **Y** from (A). Finally, complement (A).

32 *STORE A + Q*

Add (Q) to the previous content of A. Then store (A) at storage address Y as directed by the Operand Interpretation Designator, **k**.

33 *STORE A - Q*

Subtract (Q) from the previous content of A. Then store (A) at storage address Y as directed by the Operand Interpretation Designator, **k**.

34 *REPLACE Y + Q*

Clear A. Then transmit (Q) to A. Then add (Y) to (A). Then store (A) at storage address Y.

35 *REPLACE Y - Q*

Clear A. Then transmit (Q) to A. Then subtract (Y) from (A). Then complement (A) and store at storage address Y.

36 *REPLACE Y + 1*

Clear A. Then set (A) = 1. Then add (Y) to (A). Then store (A) at storage address Y.

37 *REPLACE Y - 1*

Clear A. Then set (A) = 1. Then subtract (Y) from (A). Then complement (A) and store at storage address Y.

40 *ENTER LOGICAL PRODUCT*

Enter in A the bit-by-bit product of Y and (Q).

The j designator is interpreted in a special way for this instruction for the value j = 2 or 3. If j = 2, Skip if the parity of $(A)_f$ is even. If j = 3, Skip if the parity of $(A)_f$ is odd.

NOTE:

Even parity = *an even number of "ones" in the A-register.*
Odd parity = *an odd number of "ones" in the A-register.*

41 *ADD LOGICAL PRODUCT*

Add to (A) the bit-by-bit product of Y and (Q).

42 *SUBTRACT LOGICAL PRODUCT*

Subtract from (A) the bit-by-bit product of Y and (Q).

43 *COMPARE MASKED*

Subtract from (A) the bit-by-bit product of Y and (Q), and perform the branch point evaluation for Skip of next sequential instruction as directed by the Branch Condition Designator, j.

This instruction results in no net change in the content of any operational register. It provides, through the Branch Condition Designator, j, a comparison of a portion of Y with (A).

44 *REPLACE LOGICAL PRODUCT*

Enter in A the bit-by-bit product of (Y) and (Q). Then store (A) at storage address Y.

The j designator is interpreted in a special way for this instruction for the values j = 2 or 3. If j = 2, Skip if the parity of $(A)_f$ is even. If j = 3, Skip if the parity of $(A)_f$ is odd.

NOTE:

Even parity = *an even number of "ones" in the A-register.*
Odd parity = *an odd number of "ones" in the A-register.*

45 *REPLACE A + LOGICAL PRODUCT*

Add to (A) the bit-by-bit product of (Y) and (Q). Then store (A) at storage address Y.

46 *REPLACE A - LOGICAL PRODUCT*

Subtract from (A) the bit-by-bit product of (Y) and (Q). Then store (A) at storage address Y.

47 *STORE LOGICAL PRODUCT*

Store in address Y the bit-by-bit product of (A) and (Q) as directed by the Operand Interpretation Designator, **k**.

50 *SELECTIVE SET*

Set the individual bits of A to *one* corresponding to *ones* in Y leaving the remaining bits of A unaltered.

51 *SELECTIVE COMPLEMENT*

Complement the individual bits of A corresponding to *ones* in Y leaving the remaining bits of A unaltered.

52 *SELECTIVE CLEAR*

Clear the individual bits of A corresponding to *ones* in Y leaving the remaining bits of A unaltered.

In this instruction, **k** = 7 should not be used.

53 *SELECTIVE SUBSTITUTE*

Set the individual bits of A with bits of Y corresponding to *ones* in Q leaving the remaining bits of A unaltered.

In this instruction, **k** = 7 should not be used.

54 *REPLACE SELECTIVE SET*

Set the individual bits of A to *one* corresponding to *ones* in (Y) leaving the remaining bits of A unaltered. Then store (A) at storage address Y.

55 *REPLACE SELECTIVE COMPLEMENT*

Complement the individual bits of A corresponding to *ones* in (Y) leaving the remaining bits of A unaltered. Then store (A) at storage address Y.

56 *REPLACE SELECTIVE CLEAR*

Clear individual bits of A corresponding to *ones* in (Y) leaving the remaining bits of A unaltered. Then store (A) at storage address Y.

57 *REPLACE SELECTIVE SUBSTITUTE*

Clear individual bits of A corresponding to *ones* in Q leaving the remaining bits of A unaltered. Then form the bit-by-bit product of (Y) and (Q), and set *ones* of this product in corresponding bits of A leaving the remaining bits of A unaltered. Then store (A) at storage address Y.

60 *JUMP (Arithmetic)*

This instruction clears the Program Address Register, P, and enters a new program address in P for certain conditions of either the A- or Q-register content. The Branch Condition Designator, j, is interpreted in a special way for this instruction and thus determines the conditions under which a Jump in program address occurs. If the Jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the Jump condition is satisfied, as listed below, then Y becomes the address of the next instruction and the beginning of a new program sequence.

j = 0:	No jump. Set Interrupt Enable to remove interrupt lockout, thus clearing Bootstrap and Interrupt modes. Continue with current program sequence.
j = 1:	Execute jump. Set Interrupt Enable to remove interrupt lockout, thus clearing Bootstrap and Interrupt modes.
j = 2:	Execute jump if (Q) is positive.
j = 3:	Execute jump if (Q) is negative.
j = 4:	Execute jump if (A) is zero.
j = 5:	Execute jump is (A) is nonzero.
j = 6:	Execute jump if (A) is positive.
j = 7:	Execute jump if (A) is negative.

61 *JUMP (Manual)*

This instruction clears the Program Address Register, P, and enters a new program address in P for certain conditions of manual JUMP key selections. The Branch Condition Designator, j, is interpreted in a special way for this instruction and thus determines the conditions under which a jump in program address occurs. If the Jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the Jump condition is satisfied, as listed below, then Y becomes the address of the next instruction and the beginning of a new program sequence.

Program execution may be stopped by certain STOP selections on execution of this instruction. The Branch Condition Designator, j, specifies which key selections are effective.

j = 0:	Execute jump regardless of key selections.
j = 1:	Execute jump if JUMP 1 is selected.
j = 2:	Execute jump if JUMP 2 is selected.
j = 3:	Execute jump if JUMP 3 is selected.
j = 4:	Execute jump. Stop computation.
j = 5:	Execute jump. Stop computation if STOP 5 is selected.
j = 6:	Execute jump. Stop computation if STOP 6 is selected.
j = 7:	Execute jump. Stop computation if STOP 7 is selected.

62 *JUMP ON C^n ACTIVE INPUT BUFFER*

This instruction clears the Program Address Register, P, and enters a new program address in P for certain input buffer conditions on the channel designated by $\hat{j}$. If the buffer is active, the Jump condition is satisfied; then Y becomes the address of the next instruction. If the buffer in inactive, the Jump condition is not satisfied. The next sequential instruction in the current sequence is executed in the normal manner. $\hat{k}$ = 0, 1, 2, or 3 is permitted.

63 *JUMP ON C^n ACTIVE OUTPUT BUFFER*

This instruction clears the Program Address Register, P, and enters a new address in P for certain output buffer conditions on the channel designated by $\hat{j}$. If the buffer is active, the Jump condition is satisfied; then Y becomes the address of the next instruction. If the buffer is inactive, the Jump condition is not satisfied. The next sequential instruction in the current sequence is executed in the normal manner. $\hat{k}$ = 0, 1, 2, or 3 is permitted.

64 *RETURN JUMP (Arithmetic)*

This instruction executes a Return-Jump sequence for certain conditions of either the A- or Q-register content. The Branch Condition Designator, j, is interpreted in a special way for this instruction and determines the conditions under which the Return-Jump sequence is executed. If the Return-Jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the Return-Jump condition is satisfied, as listed below, the following sequence is performed.

Store (P) + 1 in the lower half of memory address **Y**. Then jump to **Y** + 1.

j = 0:	No action; continue with the current program sequence.
j = 1:	Execute return jump.
j = 2:	Execute return jump if (Q) is positive.
j = 3:	Execute return jump if (Q) is negative.
j = 4:	Execute return jump if (A) is zero.
j = 5:	Execute return jump if (A) is nonzero.
j = 6:	Execute return jump if (A) is positive.
j = 7:	Execute return jump if (A) is negative.

65 *RETURN JUMP (Manual)*

This instruction executes a Return Jump sequence for certain conditions of manual key selections. The Branch Condition Designator, **j**, is interpreted in a special way for this instruction and determines the conditions under which the Return Jump sequence is executed. If the Return Jump condition is not satisfied, the next sequential instruction in the current sequence is executed in a normal manner. If the Return Jump condition is satisfied, as listed below, the following sequence is performed.

Store (P) + 1 in the lower half of memory address **Y**. Then jump to **Y** + 1.

j = 0:	Execute return jump regardless of key selections.
j = 1:	Execute return jump if JUMP 1 is selected.
j = 2:	Execute return jump if JUMP 2 is selected.
j = 3:	Execute return jump if JUMP 3 is selected.
j = 4:	Execute return jump. Then stop computation.
j = 5:	Execute return jump. Stop computation if STOP 5 is selected.
j = 6:	Execute return jump. Stop computation if STOP 6 is selected.
j = 7:	Execute return jump. Stop computation if STOP 7 is selected.

66 *TERMINATE C^n INPUT BUFFER*

This instruction terminates the input buffer on channel $\hat{\mathbf{j}}$. No Input Buffer Monitor Interrupt will occur.

The Operand Interpretation Designator, $\hat{\mathbf{k}}$, the Index Designator, **b**, and the Operand Designator, **Y**, bits are not translated for this instruction.

67 *TERMINATE C^n OUTPUT BUFFER*

This instruction terminates the output buffer on channel $\hat{\mathbf{j}}$. No Output Buffer Monitor Interrupt will occur.

The Operand Interpretation Designator, $\hat{k}$, the Index Designator, **b**, and the Operand Designator, **Y**, bits are not translated for this instruction.

70 *REPEAT*

Clear B^7 and transmit the lower 15 bits of **Y** to B^7. If **Y** is nonzero, transmit (**j**) to **r** (designator register), thereby initiating the *repeat mode*. If **Y** is zero, skip the next instruction.

REPEAT MODE - The *repeat mode* executes the instruction immediately following the Repeat instruction **Y** times; B^7 contains the number of executions remaining throughout the repeat mode.

If no Skip condition is met for the repeated instruction, the *repeat mode* terminates. The instruction following the repeated instruction is then executed. If the Skip condition for the repeated instruction is met, the *repeat mode* terminates, and the instruction following the repeated instruction is skipped.

Following the *repeat mode* termination, the count remains in B^7. In no way does the *repeat mode* alter a repeated instruction as stored in memory.

The three lower-order bits of the **r** designator (from **j** of instruction 70) affect operand indexing as follows:

r = 0:	Do not modify the operand address of the repeated instruction after each individual execution.
r = 1:	Increase the operand address of the repeated instruction by one after each execution of the repeated instruction.
r = 2:	Decrease the operand address of the repeated instruction by one after each execution of the repeated instruction.
r = 3:	Repeat the initial B-register modification of the repeated instruction before each execution.
r = 4:	Do not modify the operand address of the repeated instruction after each individual execution. If the repeated instruction is a Replace instruction, the operand address is incremented by (B^6) for the store portion of the Replace Instruction.
r = 5:	Increase the operand address of the repeated instruction by one after each execution of the repeated instruction. If the repeated instruction

is a Replace instruction, the operand address is incremented by (B^6) for the store portion of the Replace instruction.

r = 6: Decrease the operand address of the repeated instruction by one after each execution of the repeated instruction. If the repeated instruction is a Replace instruction, the operand address is incremented by (B^6) for the store portion of the Replace instruction.

r = 7: Repeat the initial B-register modification of the repeated instruction before each execution. If the repeated instruction is a Replace instruction, the operand address is incremented by (B^6) for the store portion of the Replace instruction.

NOTE:

Instruction 70 j *designator establishes the repeat mode* r *designator since* j *is transmitted to* r.

71 *B SKIP ON* B^n

If the content of B-register j is equal to Y, skip the next instruction in the current sequence and proceed to the instruction following. Clear B-register j.

If the content of B-register j is not equal to Y, proceed to the next instruction in the sequence in a normal manner. Increase the content of B-register j by one.

The Branch Condition Designator, j, is used to designate the selected B-register in this instruction and is not available for its normal function. Only the lower-order 15 bits of Y are used in the comparison described in the preceding paragraph.

72 *B JUMP ON* B^n

If the content of B-register j is *nonzero* execute a jump in program address to address Y. Reduce the content of B-register j by one.

If the content of B-register j is *zero*, proceed to the next instruction in a normal manner. Do not alter the content of B-register j.

The Branch Condition Designator, j, is used to designate the selected B-register in this instruction and is not available for its normal function. If the Jump condition is satisfied, then the lower-order 15 bits of Y become the address of the next instruction and the beginning of the new program sequence. The higher-order 15 bits of (Y) cannot be used in this instruction.

73 *INPUT BUFFER ON C^n (without MONITOR Mode)*

This instruction establishes an input buffer via input buffer channel $\hat{j}$ to Magnetic Core Storage with an initial storage address Y. Subsequent to this instruction, individual transfers will be executed at a rate determined by an external device. The storage address initially established by this instruction will be advanced by one preceding each individual transfer. The next current address will be maintained throughout the buffer process in the lower-order 15 bits of Magnetic Core Storage address 00100 plus $\hat{j}$. This mode will continue until it is superseded by a subsequent initiation or termination of an input buffer via the same input channel or until the higher-order half and the lower-order half of storage address 00100 plus $\hat{j}$ contain equal quantities, whichever occurs first.

This instruction is implemented as follows: If $\hat{k} = 3$, store (Y) in storage location 00100 plus $\hat{j}$. If $\hat{k} = 1$, store the lower-order 15 bits of (Y) in the lower-order half of storage location 00100 plus $\hat{j}$ leaving the higher-order half undisturbed. If $\hat{k} = 0$, store Y in the lower-order half of storage location 00100 plus $\hat{j}$ leaving the higher-order half undisturbed. Proceed to the next instruction. $\hat{k} = 2$ is not permitted.

74 *OUTPUT BUFFER ON C^n (without MONITOR Mode)*

This instruction establishes an output buffer via output buffer channel $\hat{j}$ from initial storage address Y in Magnetic Core Storage. Subsequent to this instruction, the individual transfers will be executed at a rate determined by an external device. The storage address initially established by this instruction will be advanced by one preceding each individual transfer. The next current address will be maintained throughout the buffer process in the lower-order 15 bits of Magnetic Core Storage address 00120 plus $\hat{j}$. This mode will continue until it is superseded by a subsequent initiation or termination of an output buffer via the same output channel or until the higher-order half and the lower-order half of storage address 00120 plus $\hat{j}$ contain equal quantities, whichever occurs first.

This instruction is implemented as follows: If $\hat{k} = 3$, store (Y) in storage location 00120 plus $\hat{j}$. If $\hat{k} = 1$, store the lower-order 15 bits of (Y) in the lower-order half of storage location 00120 plus $\hat{j}$ leaving the higher-order half undisturbed. If $\hat{k} = 0$, store Y in the lower-order half of storage location 00120 plus $\hat{j}$ leaving the higher-order half undisturbed. Proceed to the next instruction. $\hat{k} = 2$ is not permitted.

75 *INPUT BUFFER ON C^n (with MONITOR Mode)*

This instruction establishes an input buffer via input buffer channel $\hat{j}$ to Magnetic Core Storage with an initial storage address Y. Subsequent to this instruction, the individual transfers will be executed at a rate determined by an external device. The storage address initially established by this instruction will be advanced by one preceding each individual transfer. The next current address will be maintained throughout the buffer process in the lower-order 15 bits of Magnetic Core Storage address 00100 plus $\hat{j}$. This mode will continue until it is superseded by a subsequent initiation or termination of an input buffer via the same input channel or until the higher-order half and the lower-order half of storage address 00100 plus $\hat{j}$ contain equal quantities, whichever occurs first. Initiation of this input buffer selects the input channel $\hat{j}$ and establishes a buffer monitor on input channel $\hat{j}$. A Monitor Interrupt follows completion of the buffering operation: $(00100 + j)_u = (00100 + j)_L$.

This instruction is implemented as follows: If $\hat{k} = 3$, store (Y) in storage location 00100 plus $\hat{j}$. If $\hat{k} = 1$, store the lower-order 15 bits of (Y) in the lower-order half of storage location 00100 plus $\hat{j}$ leaving the higher-order half undisturbed. If $\hat{k} = 0$, store Y in the lower-order half of storage location 00100 plus $\hat{j}$. Proceed to the next instruction. $\hat{k} = 2$ is not permitted.

76 *OUTPUT BUFFER ON C^n (with MONITOR Mode)*

This instruction establishes an output buffer via output buffer channel $\hat{j}$ from initial storage address Y in Magnetic Core Storage. Subsequent to this instruction, the individual transfers will be executed at a rate determined by an external device. The storage initially established by this instruction will be advanced by one preceding each individual transfer. The next current address will be maintained throughout the buffer process in the lower-order 15 bits of Magnetic Core Storage address 00120 plus $\hat{j}$. This mode will continue until it is superseded by a subsequent initiation ~~or~~ termination of an output buffer via the same output channel or until the higher-order half and the lower-order half of storage address 00120 plus $\hat{j}$ contain equal quantities, whichever occurs first. Initiation of this output buffer selects the output channel $\hat{j}$ and establishes a buffer monitor on output channel $\hat{j}$. A Monitor Interrupt follows the completion of the buffering operation: $(00120 + j)_u = (00120 + j)_L$.

This instruction is implemented as follows: If $\hat{k} = 3$, store (Y) in storage location 00120 plus $\hat{j}$. If $\hat{k} = 1$, store the lower-order 15 bits of (Y) in the lower-order half of storage location 00120 plus $\hat{j}$ leaving the higher-order half undisturbed. If $\hat{k} = 0$, store Y in the lower-order half of storage location 00120 plus $\hat{j}$ leaving the higher-order half undisturbed. Proceed to the next instruction. $\hat{k} = 2$ is not permitted.

APPENDIX B

INPUT/OUTPUT SPECIFICATION
FOR THE
AN/USQ-20 UNIT COMPUTER

APPENDIX B

INPUT/OUTPUT SPECIFICATION FOR THE AN/USQ-20 UNIT COMPUTER

1. INTRODUCTION

A. *General*

Communication with the AN/USQ-20 Unit Computer is carried on in a 30-bit parallel mode. The Unit Computer is provided with 14 input channels, which are divided into 12 normal and 2 special input channels, and 14 output channels, which are divided into 12 normal and 2 special output channels. *External Function Codes* are carried over the same 30 lines as are used for output data, but the control signals used with External Function Codes are carried on different lines to indicate the nature of the signals on the 30 lines.

The two *special* input channels and two *special* output channels differ from the normal channels only in timing and control of data transfer. All input/output channels maintain the same electrical specifications; minor modification makes the special input channels identical to the normal input channels. Peripheral equipment which incorporates the features necessary for inter-computer data transfer may also use the special output channels.

Note that all references, in this Appendix of the technical note, to input or output are made from the standpoint of the computer; that is, *input* is input *to* the computer and *output* is output *from* the computer.

B. *Control Communication*

The AN/USQ-20 Unit Computer is designed to use a d-c level input/output system. Signals are d-c levels which may be changed upon interchange of control information. Signals may exist for microseconds or days, depending on the nature of the particular task.

It should be noted that the control lines are carried in the same cables as the data lines and have the same voltage levels. Hence, delay times, rise and fall times, and storage times are similar.

C. *Data and Control Signals*

Each input and each output channel has its own cable associated with it (28 cables in all). Each cable has 30 data lines plus 3 of a possible 4 control lines, as listed in Table B-1.

TABLE B-1. CONTROL SIGNALS IN INPUT AND OUTPUT CABLES

NORMAL INPUT CABLE	*SPECIAL* INPUT CABLE	*NORMAL* OUTPUT CABLE	*SPECIAL* OUTPUT CABLE
Input Data Request	Input Data Request	Output Data Request	Ready
Input Acknowledge	Input Acknowledge	Output Acknowledge	Resume
Interrupt	Interrupt (not used in inter-computer communication)	External Function	(Not Used)
(Not Used)	Input Buffer Active (used only in inter-computer communication)	(Not Used)	Input Buffer Active

Figure B-1 shows the Unit Computer receiving input from Equipment I and sending output to Equipment II. Of course in most cases, both input and output cables will be used on the same peripheral equipment. Only normal output channels are used for output to peripheral equipment. Any input channel may be used for input from peripheral equipment. Note the direction of information flow. The Data Request signals are always sent from the peripheral equipment to the computer. The Acknowledge signals are always sent from the computer to the peripheral equipment. The third set of control signals, called Interrupt in the input cable and External Function in the output cable is always sent in the same direction as data flow.

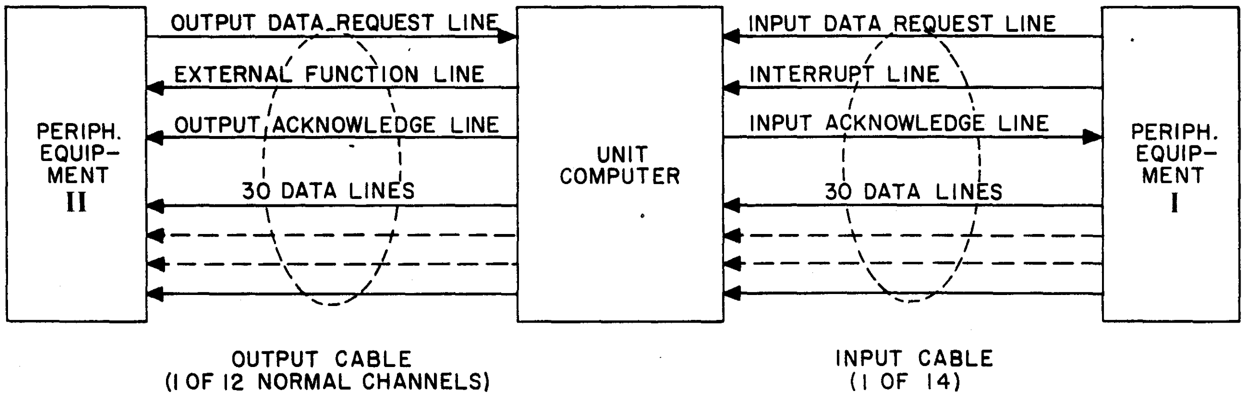


Figure B-1. Cable Connections

D. *Sequence of Events*

The sequence of events for each of four cases of communication between the AN/USQ-20 Unit Computer and peripheral equipment follows:

1) Normal Input sequence for data transfer to Unit Computer from Equipment I (Buffer mode):

   a) Computer initiates input buffer for given channel.
   b) Peripheral equipment places data word on 30 data lines.
   c) Peripheral equipment sets the Input Data Request line to indicate that it has data ready for transmission.
   d) Computer detects the Input Data Request.
   e) Computer samples the 30 data lines at its own convenience.
   f) Computer sets the Input Acknowledge line, indicating that it has sampled the data.
   g) Peripheral equipment senses the Input Acknowledge line.
   h) Peripheral equipment drops the data lines and the Input Data Request line.

   Steps b) through h) of this sequence are repeated for every data word until the number of words specified in the input buffer have been transferred.

2) Sequence for Peripheral Equipment I when transmitting an Interrupt code to computer:

   a) Peripheral equipment places the Interrupt code on the 30 data lines.
   b) Peripheral equipment sets the Interrupt line.
   c) Computer detects the Interrupt.
   d) Computer samples the 30 data lines.
   e) Computer sets the Input Acknowledge line, indicating that it has sampled the data.
   f) Peripheral equipment senses the Input Acknowledge line.
   g) Peripheral equipment drops the Interrupt code from the data lines and the Interrupt line.

   Note that the *Input Acknowledge* is the computer response to either an *Input Data Request* or to an *Interrupt*. Thus it is not permissible for a peripheral equipment to interrupt until its *Input Data Request* has been answered, since it would have no way of knowing which signal was being acknowledged.

3) Normal output sequence for data transfer from Unit Computer to Equipment II (Buffer mode):

   a) Computer initiates output buffer for given channel.
   b) Peripheral equipment sets the Output Data Request line indicating that it is in a condition to accept data.

c) Computer detects Output Data Request at its convenience.

d) Computer places information on the 30 data lines.

e) Computer sets the Output Acknowledge line, indicating that data are ready for sampling.

f) Peripheral equipment detects the Output Acknowledge.

g) Peripheral equipment may drop Output Data Request anytime after detecting Output Acknowledge.

h) Peripheral equipment samples the 30 data lines.

i) Computer drops Output Acknowledge and data lines.

Steps b) through i) of this sequence are repeated for every data word until the number of words specified in the output buffer have been transferred.

4) Sequence for Unit Computer when transmitting an External Function Code to Equipment II.

a) Computer places the External Function Code on the 30 data lines.

b) Computer sets the External Function line.

c) Peripheral equipment detects the External Function line.

d) Peripheral equipment samples the 30 data lines.

e) Computer drops External Function Code on the 30 data lines and the External Function line.

E. *Use of Special Output Channels*

Communications between two computers take place using the two special output channels reserved for this purpose. They are connected into a special input channel on the receiving computer. Note that while either a normal or a special input channel may be used for input from peripheral equipment, a *special* input channel is required for inter-computer communication.

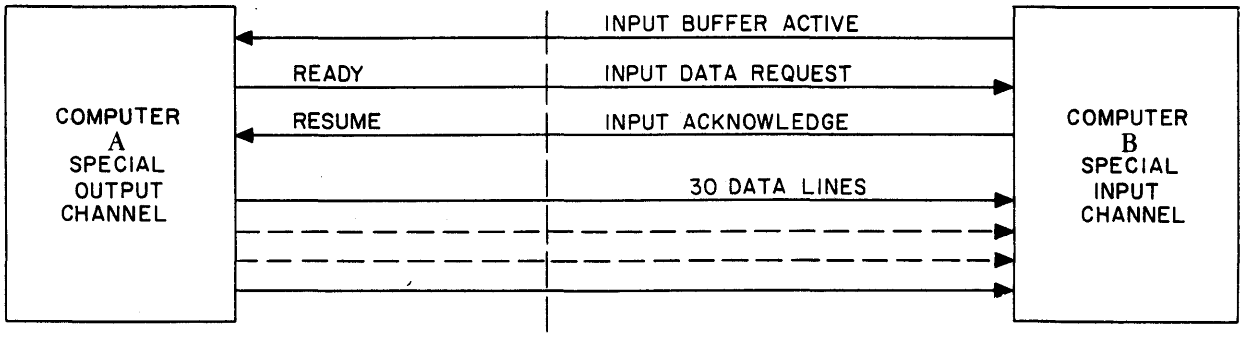


Figure B-2. Connections from Computer A to Computer B

Figure B-2 illustrates the connections for Computer A to transmit data to Computer B. Another cable using a special output channel of Computer B and a special input channel of Computer A would be necessary if Computer B were going to transmit data to Computer A.

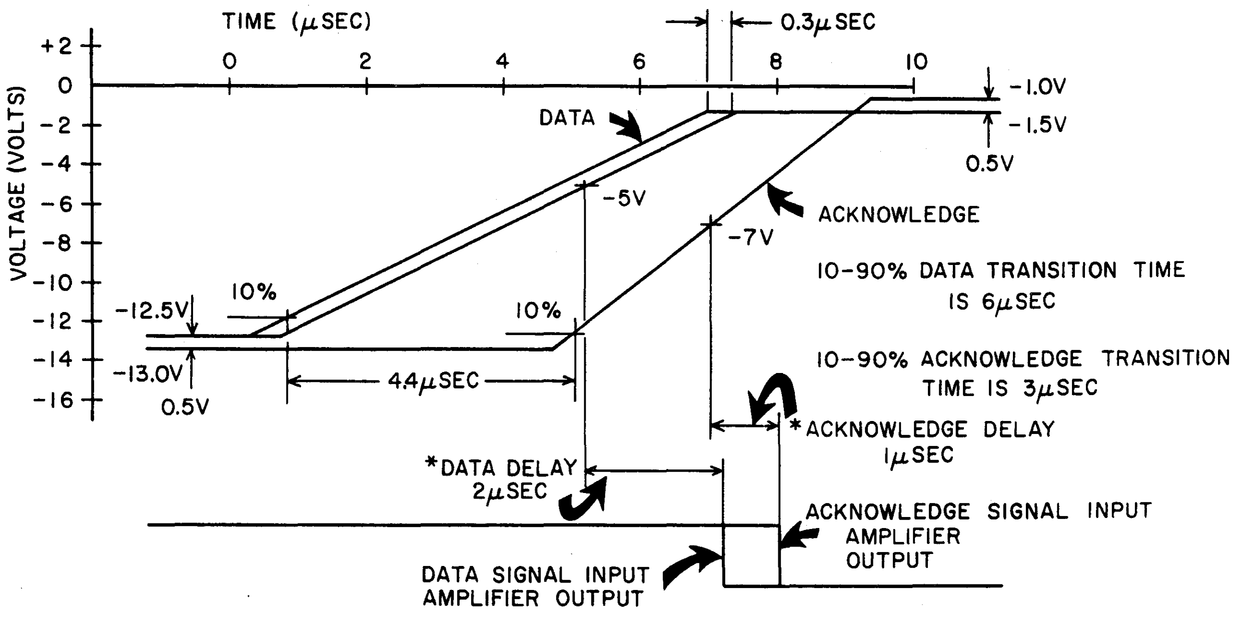
Sequence of events for normal transfer of data from Computer A to Computer B (Buffer Mode):

a) Computer B sets Input Buffer Active signal.
b) Computer A detects Input Buffer Active signal.
c) Computer A places data on 30 data lines.
d) Computer A sets *Ready* which becomes Input Data Request in Computer B.
e) Computer B detects Input Data Request.
f) Computer B samples 30 data lines.
g) Computer B sets Input Acknowledge line (returned to Computer A as *Resume*).
h) Computer A senses *Resume* line.
i) Computer A drops data lines and *Ready* line.

Steps c) through i) of this sequence are repeated for every data word. Input Buffer Active remains energized during entire transfer of block of words.

F. *Timing*

Data lines, when transmitting data from computer to equipment, *must be stable* before being sampled. Hence, a 4.4-microsecond fixed time delay exists between the computer's loading of an output register and energizing of the Acknowledge signal. Adverse tolerance build-up, causing recognition of Acknowledge signal less than a microsecond after data have reached recognition state, is illustrated in Figure B-3.

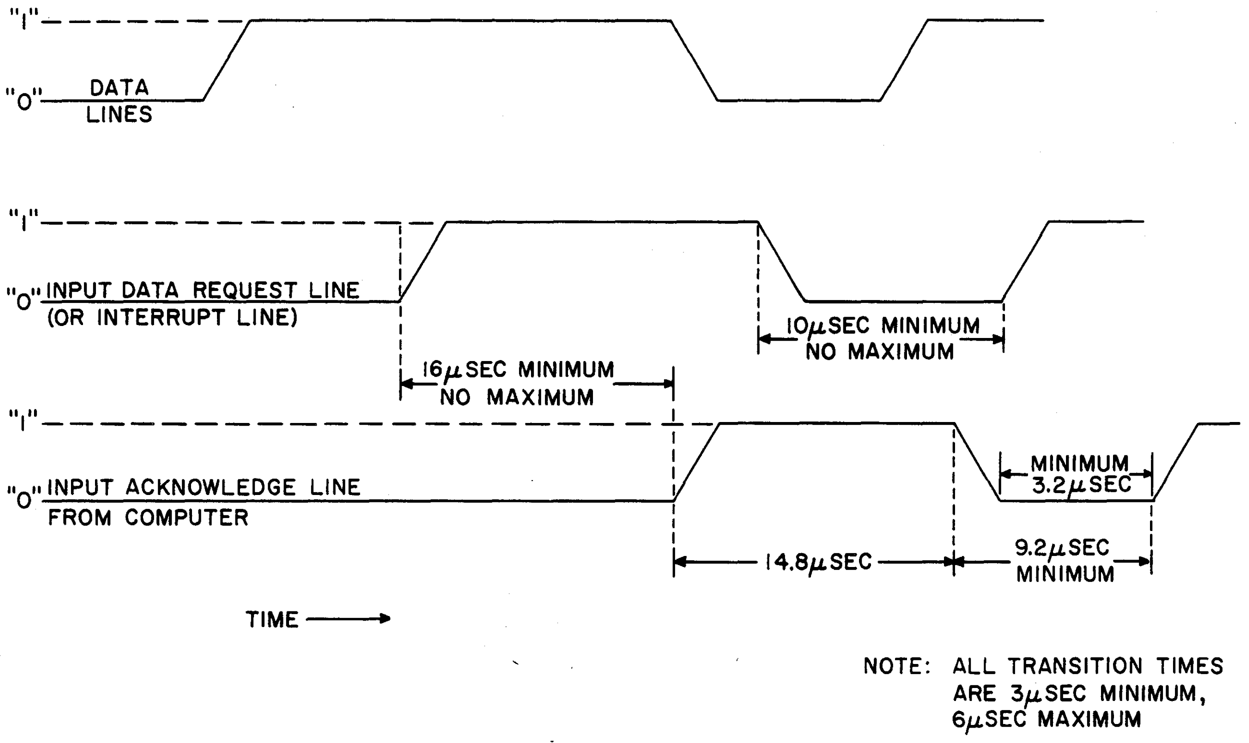


*This delay is due to integration of the input amplifier which is 1.5 μsec ± 0.5 μsec.

Figure B-3. Effect of Tolerances on Timing

1) Input Timing Considerations

The Input Data Request signal indicates to the Unit Computer that data have been placed on the 30 data lines. The Input Data Request (or Interrupt) must be maintained on the lines until an Input Acknowledge is received. Note that there is no maximum limit on the time the Input Data Request may stay up until being acknowledged. The data lines must remain stable as long as the Input Data Request is up.



"1"
"0" DATA LINES
"1"
"0" INPUT DATA REQUEST LINE (OR INTERRUPT LINE)
10μSEC MINIMUM NO MAXIMUM
16μSEC MINIMUM NO MAXIMUM
"1"
"0" INPUT ACKNOWLEDGE LINE FROM COMPUTER
MINIMUM 3.2μSEC
14.8μSEC
9.2μSEC MINIMUM
TIME
NOTE: ALL TRANSITION TIMES ARE 3μSEC MINIMUM, 6μSEC MAXIMUM

Figure B-4. Timing of Input Signals

The Input Acknowledge indicates to peripheral equipment that its 30 data lines have been sampled. The Input Acknowledge signal will be set for a fixed period of 14.8 microseconds. Peripheral equipment must be capable of detecting, as an Input Acknowledge, a signal which may exist in a stable *one* state for as little as 8.8 microseconds, allowing for the maximum permissible rise time of 6 microseconds. On sensing the Input Acknowledge, the Input Data Request (or Interrupt) may be dropped to the *zero* state anytime, but it must be dropped to the *zero* state at least 10 microseconds before another Input Data Request can be initiated. Note that the time relationships are such that peripheral equipment wishing to transmit data at a maximum rate could legitimately reset the Input Data Request before the previous Input Acknowledge had dropped to the *zero* state; however, the Input Acknowledge will always be returned to the *zero* state before being reset to the *one* state. Minimum time for which it will be dropped to the *zero* state is 9.2 microseconds. Allowing for the maximum permissible fall time of 6 microseconds, this leaves 3.2 microseconds minimum time in the *zero* state. These timing relationships are illustrated in Figure B-4.

2) Output Timing Considerations (Normal Output)

Peripheral equipment must set the Output Data Request line, indicating that it is in a condition to accept data from the Unit Computer. Data will be available to the peripheral equipment for a time interval which may be as short as 23.4 microseconds if the computer is performing output operations at a maximum rate. Data lines need not be cleared to the *zero* state before being reset to the *one* state. The time which may elapse between the request and the data being placed on the line is not fixed, but may vary from 16 microseconds upward, depending upon the computer program, the priority of the particular channel, and the data rates of the other peripheral equipment.

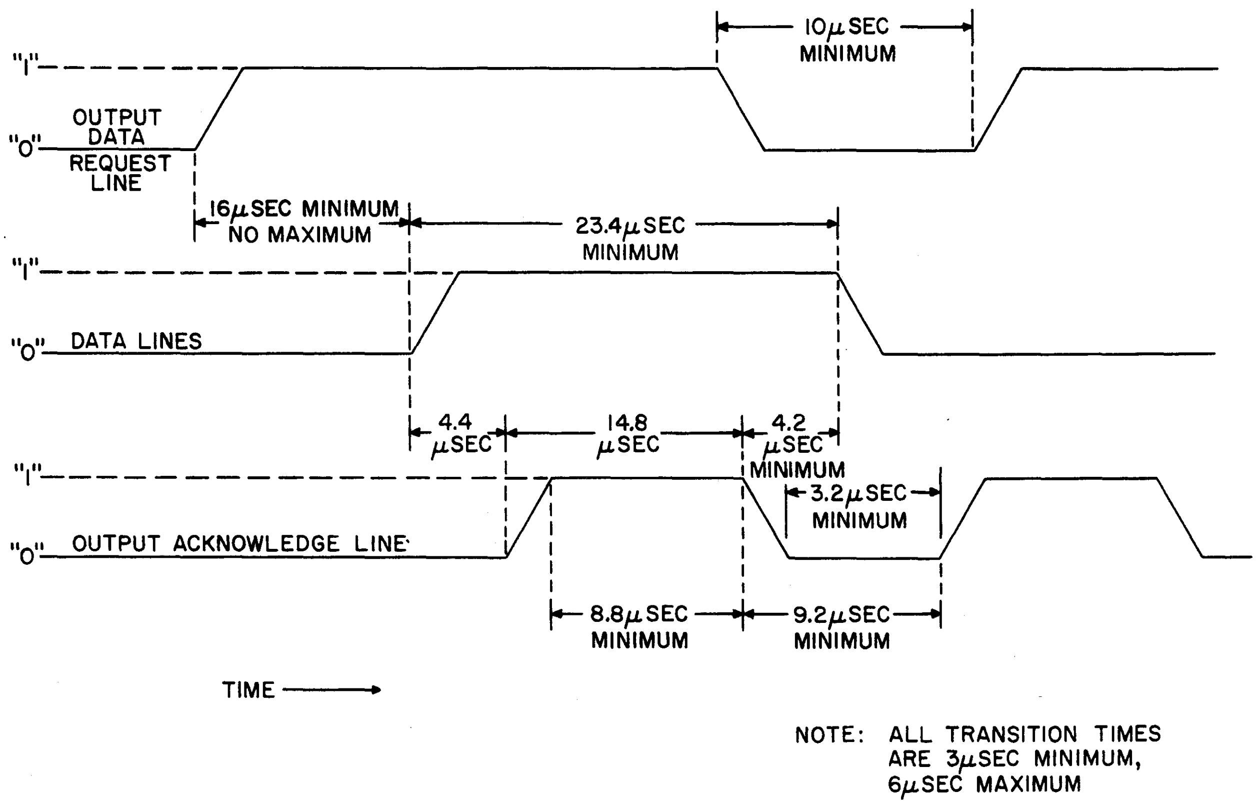


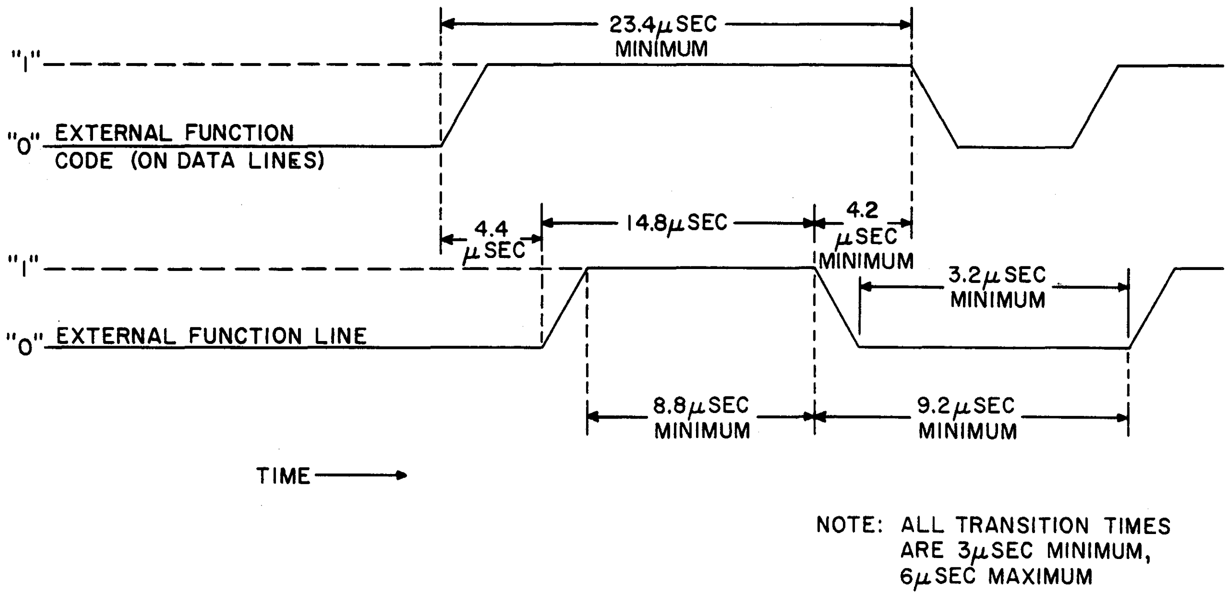
Figure B-5. Timing for Normal Output Signals

The Unit Computer will put the Output Acknowledge on the line a nominal 4.4 microseconds after placing the data on the line. Timing relationships of normal output signals are illustrated in Figure B-5.

The peripheral equipment must sample the data lines within 14.8 microseconds after the Output Acknowledge has been sent. The peripheral equipment must be capable of recognizing, as an Output Acknowledge, a signal which may exist in the stable *one* state for as little as 8.8 microseconds, allowing for a maximum permissible rise time of 6 microseconds. In view of the future desirability of speeding up the output cycle, it is recommended that all new equipment be designed to operate with an Output Acknowledge of 10 microseconds duration which would exist in a stable *one* state for a minimum of 4 microseconds.

The Unit Computer will maintain data on the lines for a minimum of 4.2 microseconds after it starts to drop the Output Acknowledge.

Output Data Request may be dropped anytime after sensing rise of the Output Acknowledge; however, the Unit Computer will not recognize another Output Data Request unless the line is dropped to the *zero* state for at least 10 microseconds. Notice that again, as in the case of input, the timing relationships allow peripheral equipment wishing to transmit data at a maximum rate to reset the Output Data Request before the Output Acknowledge for the previous request has been dropped to the *zero* state. In the worst case, Output Acknowledge will be dropped to the *zero* state for a minimum of 9.2 microseconds, which, allowing for worst case fall time, gives 3.2 microseconds in the *zero* state before being reset.



"1"
"0" EXTERNAL FUNCTION CODE (ON DATA LINES)
23.4μSEC MINIMUM
4.4 μSEC
14.8μSEC
4.2 μSEC MINIMUM
"1"
"0" EXTERNAL FUNCTION LINE
3.2μSEC MINIMUM
8.8μSEC MINIMUM
9.2μSEC MINIMUM
TIME
NOTE: ALL TRANSITION TIMES ARE 3μSEC MINIMUM, 6μSEC MAXIMUM

Figure B-6. Timing for External Function Output

3) Output Timing Considerations (External Function Output)

External Function output timing is singular in that no response is sent by the peripheral equipment. The Unit Computer places the External Function Code on the 30 data lines and follows 4.4 microseconds later with the External Function signal. The External Function remains up for a period of 14.8 microseconds. The data lines remain energized a minimum of 4.2 microseconds beyond dropping of the External Function. The External Function line will be dropped to the *zero* state for at least 9.2 microseconds before being reset. Allowing for the maximum permissible fall time of 6 microseconds, this leaves a minimum of 3.2 microseconds in the *zero*-state. Figure B-6 illustrates these timing relationships.

TECHNICAL NOTE NO. 244

DISTRIBUTION LIST

BuShips Code 687E	(100)
NEL Code 1800	(20)
NEL Code 2800	(6)
St. Paul Central File	(250)
San Diego Central File	(50)
A. P. Hendrickson	
G. G. Chapin	
L. D. Findley	
C. W. Glewwe	
R. A. Hileman	
C. J. Homan	
M. M. Koschmann	
G. E. Pickering	
J. A. Kershaw	
F. E. McLeod	
R. P. Fischer	
H. K. Smead	
T. O. Robinson	(2)
C. J. Haggerty	(2)
Contracts Department	(2)
Bureau of Ships Technical Representative - St. Paul	

Approved: F E McLeod
F. E. McLeod
Asst. Department Manager
Computer Design

Approved: J. A. Kershaw
J. A. Kershaw
Asst. Department Manager
Peripheral Equipment

Approved: G. G. Chapin
G. G. Chapin
Asst. Department Manager
Systems Development

Approved: A. P. Hendrickson
A. P. Hendrickson
Manager
Naval Tactical Data System

NTDS UNIT COMPUTER

AN/USQ-20 *Repertoire of Instructions*

JP & RJP j-DESIGNATORS

j	JP 60 / RJP 64	JP 61 / RJP 65
0	(No Jump)*	(Uncond. Jump)
1	(Uncond. Jump)*	KEY 1
2	Q POS	KEY 2
3	Q NEG	KEY 3
4	A ZERO	STOP
5	A NOT Zero	STOP 5
6	A POS	STOP 6
7	A NEG	STOP 7
$\hat{j}$	62 $\hat{j}$	63 $\hat{j}$
0-15_8	C^n ACTIVE IN	C^n ACTIVE OUT

*60 Clears interrupt & bootstrap modes.

$\hat{j}$-DESIGNATORS

(4 bits)

$\hat{j}$ Occupies 4 bit positions and represents C^n where n may be 0 — 15_8
The instruction word assumes the format:

f	$\hat{j}$	$\hat{k}$	b	y
29- -24	23 - 20	19 18	17 - 15	14- -0

$\hat{k}$-DESIGNATORS

(2 bits)

$\hat{k}$	EX—FCT 13	STR•C^n 17	JP 62 63	IN•C^n, OUT•C^n 73 75 74 76
0	'not used'	'not used'	'blank'	'blank'
1	'not used'	'not used'	L	L
2	'not used'	'not used'	U	'not used'
3	W	W	W	W

* j-DESIGNATORS

j	COM•A, •Q, •AQ 04		DIV 23	ADD•Q, SUB•Q 26 27	ENT•LP, RPL•LP 40 44	RPT 70	
0	(no skip)		(no skip)	(no skip)	(no skip)	(no mod.)	Y of NE = Y
1	(unconditional skip)		SKIP	SKIP	SKIP	ADV	Y of NE = Y+1
2	Y LESS	Y ≤ (Q)	NO Over Flow	A POS	EVEN parity	BACK	Y of NE = Y−1
3	Y MORE	Y > (Q)	Over Flow	A NEG	ODD parity	ADD B	Y of NE = Y+B^b
4	Y IN	(Q) ≥ Y and Y > (A)	A ZERO	Q ZERO	A ZERO	Rpl. Inc.	Y of NE = Y [+B^6] ✓
5	Y OUT	(Q) < Y or Y ≤ (A)	A NOT Zero	Q NOT Zero	A NOT Zero	ADV R	Y of NE = Y+1 [+B^6] ✓
6	Y LESS	Y ≤ (A)	A POS	Q POS	A POS	BACK R	Y of NE = Y−1 [+B^6] ✓
7	Y MORE	Y > (A)	A NEG	Q NEG	A NEG	ADD B R	Y of NE = Y+B^b [+B^6] ✓

✓ B^6 Increment if NI is RPL class; increments Y address for the store portion of the replace.
NE — Next execution

NORMAL j-DESIG.

j	(Not applicable on * or ^) Skip Code
0	(no skip)
1	SKIP
2	Q POS
3	Q NEG
4	A ZERO
5	A NOT Zero
6	A POS
7	A NEG

NORMAL k-DESIGNATORS

k	READ Code	READ Origin	STORE Code	STORE Dest.	REPLACE Code	REPLACE Origin	REPLACE Dest.
0	'blank'	U_L	Q	Q	'not used'	—	—
1	L	M_L	L	M_L	L	M_L	M_L
2	U	M_U	U	M_U	U	M_U	M_U
3	W	M	W	M	W	M	M
4	X	XU_L	A	A	'not used'	—	—
5	LX	XM_L	CPL	Cpl M_L	LX	XM_L	M_L
6	UX	XM_U	CPU	Cpl M_U	UX	XM_U	M_U
7	A	A	CPW	Cpl M	'not used'	—	—

LEGEND

- M – Memory word (30 bits)
- M_L – Lower half memory word
- M_U – Upper half memory word
- X – Sign bit extended
- Cpl – Complement
- A – A-register
- Q – Q-register
- U – U-register

NTDS UNIT COMPUTER

AN/USQ-20 *Repertoire of Instructions*

01	Right SHift • Q	Shift (Q) Right by Y
02	Right SHift • A	Shift (A) Right by Y
03	Right SHift • AQ	Shift (AQ) Right by Y
04*	COMpare • A, • Q, • AQ	Sense (j); $(A)_i = (A)_f$
05	Left SHift • Q	Shift (Q) Left by Y
06	Left SHift • A	Shift (A) Left by Y
07	Left SHift • AQ	Shift (AQ) Left by Y
10	ENTer • Q	$Y \rightarrow Q$
11	ENTer • A	$Y \rightarrow A$
12	ENTer • B^n	$Y \rightarrow B^j$
13^	EXternal - FunCTion • C^n	$\hat{j} \neq 0$ or 1, $(Y) \rightarrow C^{\hat{j}}$, $\hat{j}=0$ or 1, See Note.
14	SToRe • Q	$(Q) \rightarrow Y$; $k=0$, $Q' \rightarrow Q$
15	SToRe • A	$(A) \rightarrow Y$; $k=4$, $A' \rightarrow A$
16	SToRe • B^n	$(B)^j \rightarrow Y$
17^	SToRe • C^n	$(C)^{\hat{j}} \rightarrow Y$
20	ADD • A	$(A) + Y \rightarrow A$
21	SUBtract • A	$(A) - Y \rightarrow A$
22	MULtiply	$(Q)\,Y \rightarrow AQ$
23*	DIVide	$(AQ)/Y \rightarrow Q.\ R \rightarrow A_f$
24	RePLace • A+Y	$(A)+(Y) \rightarrow Y$ & A
25	RePLace • A−Y	$(A)-(Y) \rightarrow Y$ & A
26*	ADD • Q	$(Q) + Y \rightarrow Q.\ (A)_i=(A)_f$ } j interpretation
27*	SUBtract • Q	$(Q) - Y \rightarrow Q.\ (A)_i=(A)_f$ } reversed for A&Q
30	ENTer • Y+Q	$Y + (Q) \rightarrow A$
31	ENTer • Y−Q	$Y - (Q) \rightarrow A$
32	SToRe • A+Q	$(A)+(Q) \rightarrow Y$ & A
33	SToRe • A−Q	$(A)-(Q) \rightarrow Y$ & A
34	RePLace • Y+Q	$(Y)+(Q) \rightarrow Y$ & A
35	RePLace • Y−Q	$(Y)-(Q) \rightarrow Y$ & A
36	RePLace • Y+1	$(Y)+1 \rightarrow Y$ & A
37	RePLace • Y−1	$(Y)-1 \rightarrow Y$ & A
40*	ENTer • LP**	$L[Y(Q)] \rightarrow A$; j=2, even parity; j=3, odd parity
41	ADD • LP	$L[Y(Q)] + (A) \rightarrow A$
42	SUBtract • LP	$(A) - L[Y(Q)] \rightarrow A$
43	COMpare • MASK	$(A) - L[Y(Q)]$ SENSE (j), $(A)+L[Y(Q)]$; $(A)_j=(A)_f$
44*	RePLace • LP	$L(Y)(Q) \rightarrow Y$ & A; j=2, even parity; j=3, odd parity
45	RePLace • A+LP	$L(Y)(Q) + (A) \rightarrow Y$ & A
46	RePLace • A−LP	$(A) - L(Y)(Q) \rightarrow Y$ & A
47	SToRe • LP	$L(A)(Q) \rightarrow Y$; $(A)_i = (A)_f$
50	SELective • SET	SET $(A)_n$ FOR $Y_n = 1$
51	SELective • CP**	COMPLEMENT $(A)_n$ FOR $Y_n = 1$
52	SELective • CL**	CLEAR $(A)_n$ FOR $Y_n = 1$
53	SELective • SU**	$Y_n \rightarrow (A)_n$ FOR $(Q)_n = 1$
54	Replace SElective • SET	SET $(A)_n$ FOR $(Y)_n = 1$, $\rightarrow Y$ & A
55	Replace SElective • CP	COMPLEMENT $(A)_n$ FOR $(Y)_n = 1$, $\rightarrow Y$ & A
56	Replace SElective • CL	CLEAR $(A)_n$ FOR $(Y)_n = 1$, $\rightarrow Y$ & A
57	Replace SElective • SU	$(Y)_n \rightarrow (A)_n$ FOR $(Q)_n = 1$, $\rightarrow Y$
60	JumP (arithmetic)	} Jump to Y if j-condition is satisfied.
61	JumP (manual)	} (see JP & RJP j - Designators)
62^	JumP (if • C^n has ACTIVE INput buffer)	Jump to Y if $C^{\hat{j}}$ input buffer active } (see JP & RJP j - Designators)
63^	JumP (if • C^n has ACTIVE OUTput buffer)	Jump to Y if $C^{\hat{j}}$ output buffer active } (see JP & RJP j - Designators)
64	Return JumP (arithmetic)	} Jump to Y+1 and $P+1 \rightarrow Y_L$ if j condition is
65	Return JumP (manual)	} satisfied (see JP & RJP j - Designators)
66^	TERMinate • C^n • INPUT	Terminate input buffer on channel $\hat{j}$
67^	TERMinate • C^n • OUTPUT	Terminate output buffer on channel $\hat{j}$
70*	RePeaT	Execute NI Y times
71	BSKip • B^n	$(B)^j = Y$, skip NI and clear $(B)^j$; $(B)^j \neq Y$, Advance B^j and read NI
72	BJumP • B^n	$(B)^j = 0$, read NI; $(B)^j \neq 0$, $(B)^j - 1$ and jump to address Y
73^	INput • C^n (without monitor mode)	Buffer IN on $C^{\hat{j}}$; $\hat{k}=3$, $(Y) \rightarrow (00100+\hat{j})$; $\hat{k}=1$, $(Y)_L \rightarrow (00100+\hat{j})_L$; $\hat{k}=0$, $Y \rightarrow (00100+\hat{j})_L$.
74^	OUTput • C^n (without monitor mode)	Buffer OUT on $C^{\hat{j}}$; $\hat{k}=3$, $(Y) \rightarrow (00120+\hat{j})$; $\hat{k}=1$, $(Y)_L \rightarrow (00120+\hat{j})_L$; $\hat{k}=0$, $Y \rightarrow (00120+\hat{j})_L$.
75^	INput • C^n (with • MONITOR mode)	Buffer IN on $C^{\hat{j}}$ with mon. $\hat{k}=3$, $(Y) \rightarrow (00100+\hat{j})$; $\hat{k}=1$, $(Y)_L \rightarrow (00100+\hat{j})_L$; $\hat{k}=0$, $Y \rightarrow (00100+\hat{j})_L$. mon. inter. at $00040+\hat{j}$
76^	OUTput • C^n (with • MONITOR mode)	Buffer OUT on $C^{\hat{j}}$ with mon. $\hat{k}=3$, $(Y) \rightarrow (00120+\hat{j})$; $\hat{k}=1$, $(Y)_L \rightarrow (00120+\hat{j})_L$; $\hat{k}=0$, $Y \rightarrow (00120+\hat{j})_L$. mon. inter. at $00060+\hat{j}$

— NO - OPeration	} CS-1 Mono - codes
— ComPlement • A or • Q	} CS-1 Mono - codes
— CLear • A, • Q, • B^n, or Y	} CS-1 Mono - codes
— Remove Interrupt Lockout	} CS-1 Mono - codes
— Remove Interrupt Lockout and JumP • Y	} CS-1 Mono - codes
— TEST • CO or • CI	} CS-1 Mono - codes

**LP - Logical Product CP - Complement SU - Substitute CL - Clear *, ^ } Special j & k Designators (see opposite side of card) Y - The operand; Y or (Y)

NOTE: Skip NI if other Computer (on channel 0 or 1) has input buffer active. Execute twice.